تطبيق متعدد أنظمة التشغيل

.NET Multi-platform App UI (MAUI)

إنشاء تطبيقات متعددة الأنظمة باستخدام .NET MAUI

تطبيقات الهواتف وسطح المكتب

<The author/>

مجتمع المبرمجين العرب

.NET MAUI. (واجهة تطبيقات متعددة المنصات) هو إطار عمل يتيح لك إنشاء تطبيقات تعمل على أنظمة تشغيل متعددة مثل Windows, macOS, iOS, Android باستخدام قاعدة تعليمات برمجية واحدة بلغة C# ونظام مشروع واحد. يمكنك من خلاله بناء تطبيقات أصلية (Native) بواجهة مستخدم متسقة عبر جميع الأجهزة والمنصات.

مواصفات .NET MAUI.

١- تطوير متعدد الأنظمة: يمكنك إنشاء تطبيقات تعمل على أنظمة تشغيل متعددة باستخدام قاعدة تعليمات برمجية واحدة.

٢- استخدام XAML and C#: يمكنك كتابة التعليمات باستخدام لغة البرمجة C# وتصميم واجهات المستخدم باستخدام XAML

٣- تكامل مع Visual Studio 2022: يمكنك استخدام Visual Studio 2022 لتطوير وتصحيح التطبيقات.

٤- دعم مكونات واجهة المستخدم: يحتوي على مجموعة واسعة من مكونات واجهة المستخدم التي يمكنك استخدامها في تطبيقاتك.

٥- أداء عالي: يوفر أداءً عاليًا بفضل استخدامه لمكتبات .NET الأساسية.

استخدامات .NET MAUI.

١- تطوير تطبيقات الهواتف المحمولة: يمكنك إنشاء تطبيقات تعمل على Android وiOS

٢- تطوير تطبيقات سطح المكتب: يمكنك إنشاء تطبيقات تعمل على Windows وmacOS

٣- تطوير تطبيقات الأعمال: يمكنك إنشاء تطبيقات مخصصة للأعمال والشركات.

٤- تطوير تطبيقات الألعاب: يمكنك إنشاء ألعاب بسيطة باستخدام .NET MAUI.

٥- تطوير تطبيقات التعليم: يمكنك إنشاء تطبيقات تعليمية وتدريبية.

تعرف على كيفية استخدام .NET MAUI. لإنشاء تطبيقات تعمل على الهواتف الذكية والأجهزة المحمولة وعلى سطح المكتب باستخدام Visual Studio and C# ستتعلم أساسيات إنشاء تطبيق باستخدام .NET MAUI. والموضوعات الأكثر تقدماً مثل تخزين البيانات المحلية، واستدعاء خدمات الويب المستندة إلى REST

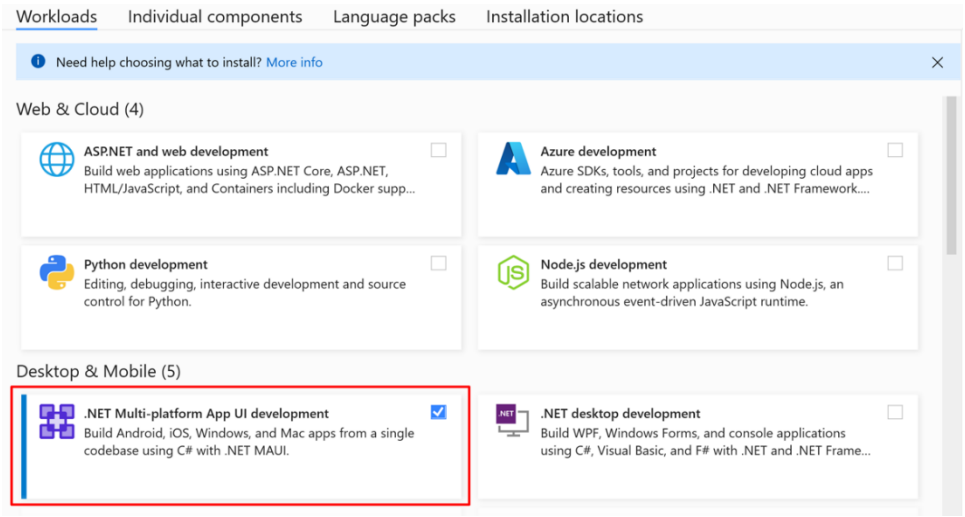
المتطلبات الأساسية:

- Visual Studio 2022 مع تثبيت حزمة عمل .NET MAUI.

لن يعمل إطار عمل .NET MAUI على الإصدارات الأقدم من Visual Studio 2022

- معرفة بأساسيات .NET, C#.
- اختياري: برنامج Visual Studio Code مع ملحق .NET MAUI and .NET SDK. مع تثبيت إطار عمل .NET MAUI.
- إذا كنت تريد إنشاء تطبيقات الويب .NET MAUI Blazor apps. يجب تثبيت إطار عمل ASP.NET and web development

راجع الوثائق للحصول على معاينة للإعداد.



التطوير باستخدام Visual Studio Code

إذا كنت تقوم بالتطوير على macOS أو Linux فستحتاج إلى تثبيت Visual Studio Code and the .NET MAUI Extension جنبا إلى جنب مع .NET SDK وإطار عمل .NET MAUI. راجع الوثائق للحصول على معاينة للإعداد.

محتويات الكتاب:

الوحدة الأولي: - إنشاء تطبيق متعدد الأنظمة باستخدام .NET MAUI.

الوحدة الثانية: - إنشاء واجهة مستخدم في تطبيق .NET MAUI. باستخدام XAML

الوحدة الثالثة: - تخصيص التخطيط layout في صفحات .NET MAUI XAML.

الوحدة الرابعة: - تصميم صفحات .NET MAUI XAML. متناسقة باستخدام الموارد المشتركة

الوحدة الخامسة: - إنشاء تطبيقات .NET MAUI. متعددة الصفحات باستخدام التنقل بين علامات التبويب والقوائم المنبثقة tab and flyout navigation

الوحدة السادسة: - إنشاء واجهة مستخدم UI تستخدم ربط البيانات data binding في .NET MAUI.

الوحدة السابعة: - استخدام خدمات ويب REST في تطبيقات .NET MAUI.

الوحدة الثامنة: - تخزين البيانات المحلية باستخدام SQLite في تطبيق .NET MAUI

الوحدة التاسعة: - تصميم نموذج عرض MVVM viewmodel لـ .NET MAUI.

الوحدة الأولي

إنشاء تطبيق متعدد الأنظمة باستخدام .NET MAUI

تعرف على كيفية استخدام Visual Studio 2022 مع .NET MAUI. لإنشاء تطبيق متعدد الأنظمة أو المنصات.

الأهداف التعليمية:

خلال هذه الوحدة، سوف تتمكن مما يلي:

- تعرف على البنية الأساسية لـ .NET MAUI.
- إنشاء تطبيق .NET MAUI.
- تحديد واجهة مستخدم مشتركة لأنظمة التشغيل المدعومة من .NET MAUI.
- نشر تطبيق .NET MAUI. من Visual Studio
- الاتصال برقم من داخل التطبيق.

محتويات الوحدة:

١- مقدمة

٢- وصف بنية NET MAUI.

٣- إنشاء مشروع NET MAUI. في Visual Studio 2022

٤- تمرين، إنشاء تطبيق NET MAUI. الأول

٥- إضافة عناصر تحكم مرئية إلى تطبيق NET MAUI.

٦- تمرين، إنشاء تطبيق مترجم رقم الهاتف

٧- الملخص

١ المقدمة

NET MAUI. هو إطار عمل متعدد المنصات multi-platform framework لإنشاء تطبيقات أصلية لسطح المكتب والهواتف native desktop and mobile apps باستخدام C# و XAML اختصار (Extensible Application Markup Language)

NET MAUI. واجهة مستخدم التطبيق متعدد الأنظمة او منصات التشغيل (Multi-platform Application User Interface) يمكنك تصميم تطبيقات الهواتف التي يمكن تشغيلها أيضاً على Windows, Android, iOS, iPadOS, and macOS

لنفترض أنك تعمل في سلسلة متاجر بقالة. تريد السلسلة توسيع برنامجها باستخدام تطبيق للهواتف وسطح المكتب. يسمح التطبيق الجديد بالاتصال بالمتجر بلمسة واحدة، كما يعرض إعلامات منبثقة حول العروض الخاصة عندما يكون المستخدم في المتجر. لذلك يحتاج التطبيق إلى الوصول إلى بعض ميزات الأجهزة.

أنت مكلف بتحديد التكنولوجيا وبناء المفهوم. يمكنك تحديد NET MAUI. كخيار تقني. يمكنك NET MAUI. من إعادة استخدام نفس التعليمات البرمجية للتطبيق والعلامات markup لإنشاء واجهة مستخدم user interface (UI) والوصول بسهولة إلى الميزات الخاصة بالأجهزة والنظام الأساسي، مثل طلب الهاتف وخدمات الموقع. بالإضافة إلى ذلك، باستخدام NET MAUI. يمكنك الاستفادة من مهارات مطورين C# الذين تعمل معهم بالفعل.

٢ وصف بنية .NET MAUI

النمط الشائع المستخدم في تطوير التطبيقات متعددة الأنظمة cross-platform هو فصل منطق العمل business logic عن واجهة المستخدم user interfaces ثم تطوير واجهات مستخدم ومنطق واجهة مستخدم UI logic منفصلين لكل نظام أساسي، بينما يظل منطق العمل دون تغيير لكل نوع من الأجهزة، فإن التعليمات التي تحرك التطبيق وتعرض البيانات يمكن أن تختلف. ويرجع هذا التباين إلى اختلاف القدرات، وواجهات برمجة التطبيقات، والميزات التي توفرها الأجهزة. يتضمن بناء تطبيق متعدد الأنظمة بهذه الطريقة التعامل ليس فقط مع مجموعات تطوير برمجيات منفصلة (SDKs) ولكن أيضًا مع لغات وأدوات مختلفة تمامًا.

الغرض من .NET MAUI. واجهة مستخدم التطبيق متعددة الأنظمة (Multi-platform Application User Interface) هو تبسيط تطوير التطبيقات متعددة الأنظمة، باستخدام .NET MAUI. يمكنك إنشاء تطبيقات متعددة النماذج باستخدام مشروع واحد، وأيضا إضافة التعليمات البرمجية، وموارد خاصة بأحد الأنظمة او المنصات المستهدفة، إذا لزم الأمر، الهدف الرئيسي من .NET MAUI. هو تمكينك من تنفيذ أكبر قدر ممكن من منطق التطبيق application logic وتخطيط واجهة المستخدم UI layout في قاعدة تعليمات برمجية واحدة.

في هذه الدرس، يمكنك التعرف على بنية .NET MAUI. والأدوات المطلوبة لإنشاء تطبيقات .NET MAUI.

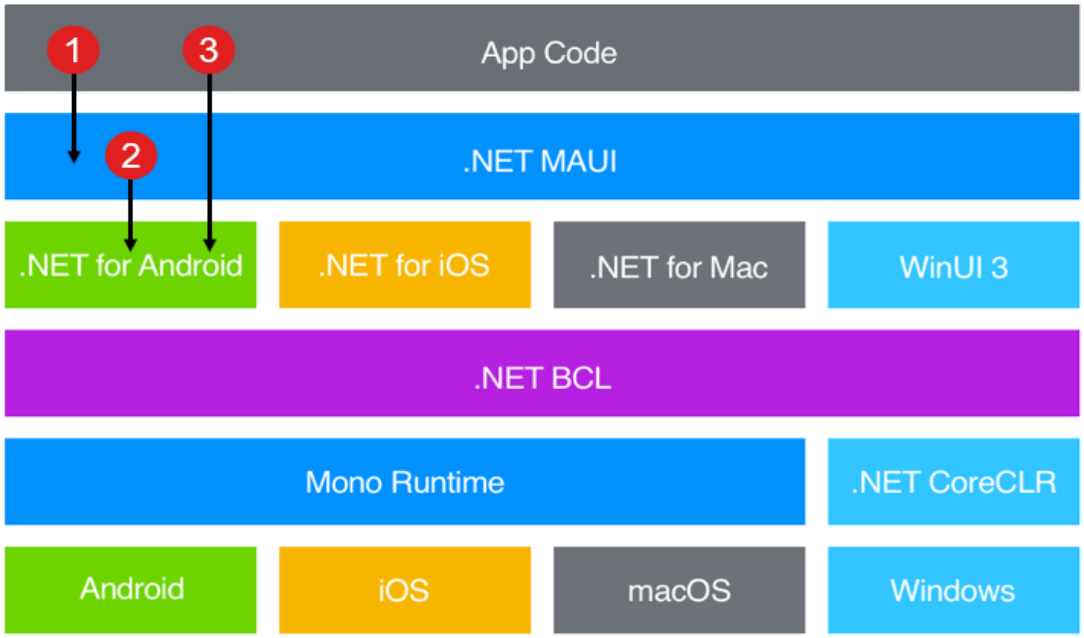
ما هي مجموعة تقنيات .NET MAUI.

توفر .NET. سلسلة من الأطر الخاصة بالأنظمة المستهدفة لإنشاء التطبيقات مثل: .NET for Android, .NET for iOS (and iPadOS), .NET for Mac, and WinUI3 باستخدام (SDK) تتمتع جميع هذه الأطر بإمكانية الوصول إلى نفس مكتبة فئات .NET 6. الأساسية (Base Class Library (BCL توفر هذه المكتبة وظيفة لإنشاء الموارد وإدارتها، ولتلخيص تفاصيل الجهاز الأساسي بشكل عام بعيدًا عن التعليمات البرمجية، يعتمد BCL على وقت تشغيل .NET. لتوفير بيئة التنفيذ للتعليمات البرمجية.

Mono هو تطبيق مفتوح المصدر لوقت تشغيل .NET runtime. ينفذ بيئات Android, iOS (and iPadOS), and macOS وفي Windows يؤدي Win32 نفس الدور، باستثناء أنه محسن للنظام الأساسي ل Windows

في حين أن BCL تمكن التطبيقات التي تعمل على أنواع مختلفة من الأجهزة من مشاركة منطق العمل المشترك، فإن الانظمة المختلفة لديها طرق مختلفة لتحديد واجهة مستخدم التطبيق. توفر الأنظمة نماذج مختلفة لتحديد كيفية تواصل عناصر واجهة المستخدم والتفاعل فيما بينها. يمكنك تصميم واجهة المستخدم لكل منصة على حدة باستخدام إطار العمل المناسب لكل نظام تشغيل (.NET for Android, .NET for iOS, .NET for Mac, or WinUI 3) لكن هذا النهج يتطلب منك الاحتفاظ بقاعدة تعليمات برمجية لكل عائلة أجهزة فردية، يوفر .NET MAUI. إطار عمل واحد لبناء واجهات المستخدم للتطبيقات الهواتف وسطح المكتب، يمكنك إنشاء واجهة المستخدم باستخدام إطار العمل هذا (المشار إليه بواسطة السهم ١ في الرسم التخطيطي التالي)، ويتولى .NET MAUI. بتحويله إلى النظام الأساسي المناسب (السهم ٢)

قد تكون هناك أوقات تحتاج فيها إلى تنفيذ ميزة خاصة بنظام تشغيل محدد، في هذه الحالات، يمكنك استدعاء الأساليب في إطار العمل الخاص بالنظام، كما هو موضح في السهم ٣ في الرسم التخطيطي التالي.



كيف يعمل .NET MAUI؟

يقوم .NET MAUI بتجريد تنفيذ عنصر واجهة المستخدم من وصفه المنطقي، يمكنك وصف واجهة المستخدم باستخدام XAML لغة ترميز التطبيقات القابلة للتوسيع (Extensible Application Markup Language) وهي لغة محايدة للمنصة تستند إلى XML على سبيل المثال، يوضح مقطع XAML التالي وصف عنصر تحكم زر Button:

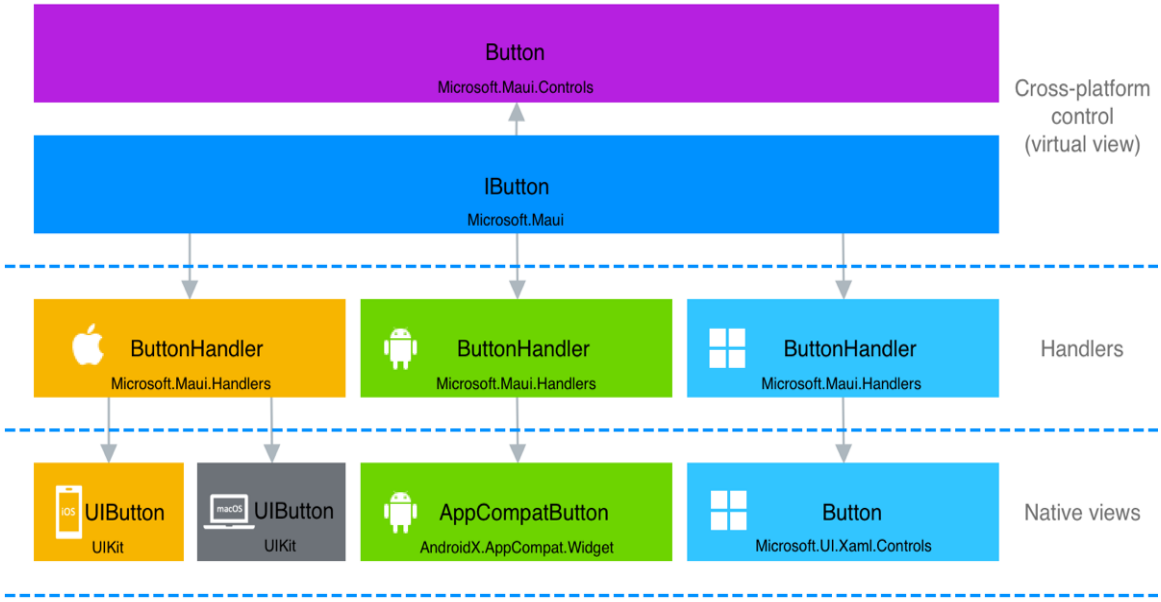
```
<Button Text="Click me"
        SemanticProperties.Hint="Counts the number of
times you click"
        Clicked="OnCounterClicked"
        HorizontalOptions="Center" />
```

ت

يعرف هذا المثال تسمية الزر `"Click me"` ويحدد أنه يجب تشغيل أسلوب مسمى `OnCounterClicked` عندما يضغط المستخدم الزر، يمكن للخصائص الأخرى تعديل تخطيط الزر والنص؛ في هذا المثال، يتم توسيط النص على الزر بخاصية `HorizontalOptions` توفر الخصائص الدلالية `SemanticProperties` الدعم لإمكانية الوصول للمستخدمين الذين يعانون من ضعف بصري.

يقوم .NET MAUI دائما بإنشاء تعليمات برمجية أصلية للجهاز المستهدف، حتى تحصل على الأداء الأمثل، يستخدم .NET MAUI معالجات خاصة ومحددة لكل نظام تشغيل، وكل عنصر من عناصر واجهة المستخدم لتنفيذ عملية. على سبيل المثال،

إذا كنت تستهدف نظام التشغيل iOS لهذا التطبيق، فإن معالج .NET MAUI يربط هذه التعليمة البرمجية بـ iOS UIButton إذا كنت تعمل على نظام التشغيل Android فستحصل على Android AppCompatButton يمكن الوصول إلى هذه المعالجات بشكل غير مباشر من خلال واجهة تحكم خاصة control-specific interface يوفرها .NET MAUI. مثل IButton لزر button



ملاحظة

إذا كنت تفضل ذلك، يمكنك أيضا إنشاء واجهة المستخدم ديناميكياً باستخدام التعليمات البرمجية في C# يمكنك هذا الأسلوب من تعديل التخطيط وفقاً للبيئة، على سبيل المثال، قد لا ترغب في ظهور عناصر تحكم معينة إذا لم يكن لدى المستخدم مستوى تخويل مناسب.

.NET MAUI يسهل الوصول إلى عناصر التحكم الشائعة مثل الأزرار، كما أن عناصر التحكم الشائعة الأخرى - مثل حقول إدخال النص والتسميات ومحددات التاريخ - هي بنفس السهولة، ومع ذلك، فإن عناصر التحكم الفردية ليست كافية لإنشاء نظام أساسي جيد لإنشاء تطبيقات غنية.

يوفر .NET MAUI أيضاً:

- محرك تخطيط مفصل ومتقن لتصميم الصفحات.
- أنواع متعددة من الصفحات لإنشاء أنواع تنقل غنية، مثل الدرج.
- دعم ربط البيانات، لأنماط تطوير أكثر أناقة وقابلة للصيانة.

- القدرة على إنشاء معالجات مخصصة لتحسين الطريقة التي يتم بها عرض عناصر واجهة المستخدم.
- الوصول إلى واجهات برمجة التطبيقات الأصلية مباشرة، وتجريد العديد من الاحتياجات الشائعة لتطبيقات الجوال وسطح المكتب mobile & desktop المنفصلة عن واجهة المستخدم، تتيح مكتبة الأساسيات للتطبيق الوصول إلى أشياء مثل نظام تحديد المواقع العالمي (GPS) ومقياس التسارع وحالة البطارية والشبكة. هناك عشرات من أجهزة الاستشعار والخدمات المشتركة في تطوير الهواتف متاحة أيضا من خلال هذه المكتبة.

اختبار بسيط:

١ ما هي البيئة التي توفر دعم وقت التشغيل لتطبيق WinUI 3

٢ ما هي لغة العلامات التي يمكنك استخدامها لتخطيط واجهة المستخدم لتطبيق .NET MAUI

حل الاختبار:

١ يوفر Win32 دعم وقت التشغيل لتطبيقات WinUI 3 التي تم إنشاؤها باستخدام .NET MAUI

٢ XAML (Extensible Application Markup Language) هي اللغة التي تستخدمها لتخطيط واجهة مستخدم تطبيق .NET MAUI

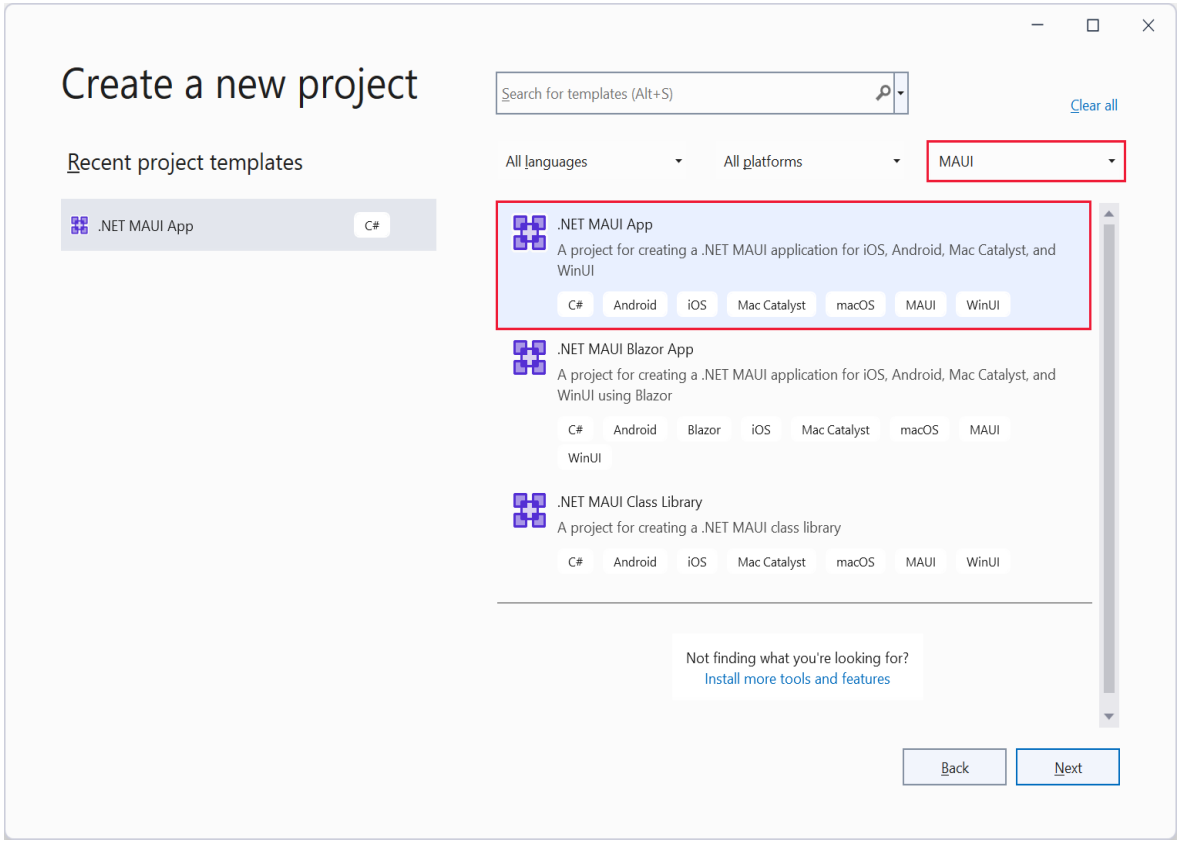
٣ إنشاء مشروع .NET MAUI في Visual Studio 2022

بعد تثبيت أدوات .NET MAUI. وتكوينها، يمكنك استخدام Visual Studio لإنشاء تطبيق .NET MAUI. واجهة مستخدم التطبيق متعدد الأنظمة Multi-platform Application User Interface

في هذا الدرس، ستتعرف على بنية قالب .NET MAUI في Visual Studio واستخدام هذا القالب لإنشاء تطبيقات الهواتف وسطح المكتب عبر أنظمة التشغيل.

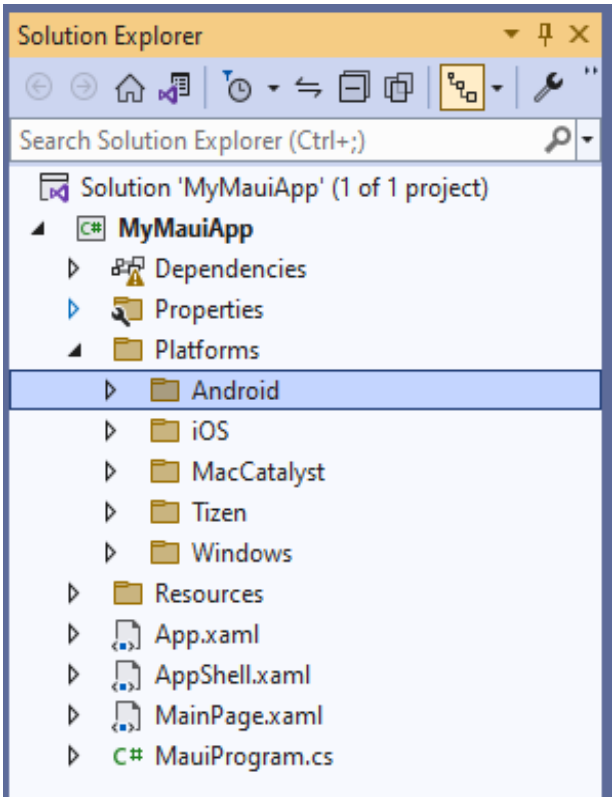
كيفية البدء

لإنشاء مشروع .NET MAUI. جديد، في مربع الحوار إنشاء مشروع جديد Create a new project حدد نوع مشروع .NET MAUI. ثم اختر قالب تطبيق .NET MAUI



اتبع الخطوات الواردة في المعالج لتسمية المشروع، وتحديد موقع المشروع.

يحتوي مشروع .NET MAUI الذي تم إنشاؤه على العناصر التالية، كما هو موضح:



بنية مشروع .NET MAUI. وبدء تشغيل التطبيق

تتضمن محتويات المشروع العناصر التالية:

App.xaml يعرف هذا الملف موارد التطبيق التي يستخدمها التطبيق في تخطيط XAML مثل الألوان والخطوط، توجد الموارد الافتراضية في المجلد Resources وتحدد الألوان على مستوى التطبيق، والأنماط الافتراضية لكل عنصر تحكم مضمن في .NET MAUI. هنا، ترى قاموسي الموارد resource dictionaries يتم دمجهما معا:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:MyMauiApp"
             x:Class="MyMauiApp.App">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
               <ResourceDictionary Source="Resources/Colors.xaml" />
               <ResourceDictionary Source="Resources/Styles.xaml" />
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

App.xaml.cs هذا الملف هو التعليمات البرمجية بالخلفية، الخاصة بملف App.xaml وهو يعرف فئة التطبيق App class تمثل هذه الفئة تطبيقك في وقت التشغيل، تقوم الدالة الإنشائية The constructor في هذه الفئة بإنشاء نافذة أولية، وتعيينها لخاصية MainPage property تحدد هذه الخاصية الصفحة التي يتم عرضها أولاً، عند بدء تشغيل التطبيق.

بالإضافة إلى ذلك، تتيح لك هذه الفئة تجاوز معالجات أحداث دورة حياة التطبيق المحايدة الشائعة لنظام التشغيل، تتضمن الأحداث OnStart, OnResume, OnSleep يتم تعريف هذه المعالجات كأعضاء في فئة التطبيق الأساسية Application base class توضح التعليمات البرمجية التالية أمثلة:

ملاحظة:

يمكنك تجاوز أحداث دورة الحياة الخاصة بنظام التشغيل، عند بدء تشغيل التطبيق لأول مرة، يتم وصف ذلك لاحقاً.

```
namespace MyMauiApp;

public partial class App : Application
{
    public App()
    {
        InitializeComponent();

        MainPage = new AppShell();
    }

    protected override void OnStart()
    {
        base.OnStart();
    }

    protected override void OnResume()
    {
        base.OnResume();
    }

    protected override void OnSleep()
    {
        base.OnSleep();
    }
}
```

AppShell.xaml هذا الملف هو البنية الرئيسية main structure لتطبيق .NET MAUI يوفر .NET MAUI Shell العديد من الميزات المفيدة للتطبيقات متعددة الأنظمة، بما في ذلك تصميم التطبيق، والتنقل المستند إلى URI وخيارات التخطيط، بما في ذلك التنقل المنبثق flyout navigation وعلامات التبويب لجذر التطبيق tabs for the application's root يوفر القالب الافتراضي صفحة واحدة، أو (ShellContent) يتم تضخيمها عند بدء تشغيل التطبيق.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <Shell
      x:Class="MyMauiApp.AppShell"
      xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
      xmlns:local="clr-namespace:MyMauiApp"
      Shell.FlyoutBehavior="Disabled">

      <ShellContent
          Title="Home"
          ContentTemplate="{DataTemplate local:MainPage}"
          Route="MainPage" />

  </Shell>
```

MainPage.xaml يحتوي هذا الملف على تعريف واجهة المستخدم، يحتوي نموذج التطبيق الافتراضي الذي ينشئه قالب تطبيق MAUI على ملصقين وزر وصورة، يتم ترتيب عناصر التحكم باستخدام عنصر VerticalStackLayout محاط بعنصر ScrollView

يقوم عنصر VerticalStackLayout بترتيب عناصر التحكم عمودياً (في مجموعة) ويوفر ScrollView شريط تمرير، إذا كانت محتويات العرض كبيرة جدًا بحيث لا يمكن عرضها على شاشة الجهاز.

من المفترض أن تقوم باستبدال محتويات هذا الملف بتخطيط واجهة المستخدم الخاص بك، يمكنك أيضًا تعريف المزيد من صفحات XAML إذا كان لديك تطبيق متعدد الصفحات.

```
<?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="MyMauiApp.MainPage">

    <ScrollView>
        <VerticalStackLayout
            Spacing="25"
```

```
            Padding="30,0"
            VerticalOptions="Center">

            <Image
                Source="dotnet_bot.png"
                SemanticProperties.Description="Cute dot net bot
waving hi to you!"
                HeightRequest="200"
                HorizontalOptions="Center" />

            <Label
                Text="Hello, World!"
                SemanticProperties.HeadingLevel="Level1"
                FontSize="32"
                HorizontalOptions="Center" />

            <Label
                Text="Welcome to .NET Multi-platform App UI"
                SemanticProperties.HeadingLevel="Level2"
                SemanticProperties.Description="Welcome to dot net
Multi platform App U I"
                FontSize="18"
                HorizontalOptions="Center" />

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

        </VerticalStackLayout>
    </ScrollView>

</ContentPage>
```

MainPage.xaml.cs يحتوي هذا الملف على التعليمات البرمجية الخلفية للصفحة. في هذا الملف، يمكنك تعريف المنطق لمعالجات الأحداث event handlers المختلفة، والإجراءات الأخرى التي يتم تشغيلها بواسطة عناصر التحكم على الصفحة.

تنفذ التعليمات البرمجية المثال معالج لحدث النقر Clicked للزر الموجود على الصفحة، تقوم التعليمات ببساطة بزيادة متغير عداد، وتعرض النتيجة في ملصق على الصفحة.

تدعم الخدمة الدلالية المقدمة كجزء من مكتبة MAUI Essentials إمكانية الوصول، يحدد الأسلوب الثابت `Announce` method للفئة SemanticScreenReader class النص الذي أعلن عنه قارئ الشاشة عندما يحدد المستخدم الزر:

```
namespace MyMauiApp;

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);
    }
}
```

MauiProgram.cs يحتوي كل نظام تشغيل أصلي native platform على نقطة بداية مختلفة تقوم بإنشاء التطبيق وتهيئته، يمكنك العثور على هذه التعليمة البرمجية في مجلد Platforms في المشروع، هذه التعليمة البرمجية خاصة بنظام التشغيل الأساسي، ولكن في النهاية يستدعي الأسلوب CreateMauiApp method الفئة الثابتة static MauiProgram class يستخدم الأسلوب CreateMauiApp لتكوين التطبيق عن طريق إنشاء كائن منشئ التطبيق app builder object على الأقل، تحتاج إلى تحديد الفئة التي تصف تطبيقك باستخدام الأسلوب العام UseMauiApp لكائن منشئ التطبيق؛ تحدد معلمة النوع the type parameter فئات التطبيق application class (<App>)

يوفر منشئ التطبيق أيضًا طرقًا لمهام مثل تسجيل الخطوط، وتكوين الخدمات لإدخال التبعيات، وتسجيل المعالجات المخصصة لعناصر التحكم والمزيد، تعرض التعليمات البرمجية التالية مثالاً على استخدام منشئ التطبيق لتسجيل خط:

```
namespace MyMauiApp;
```

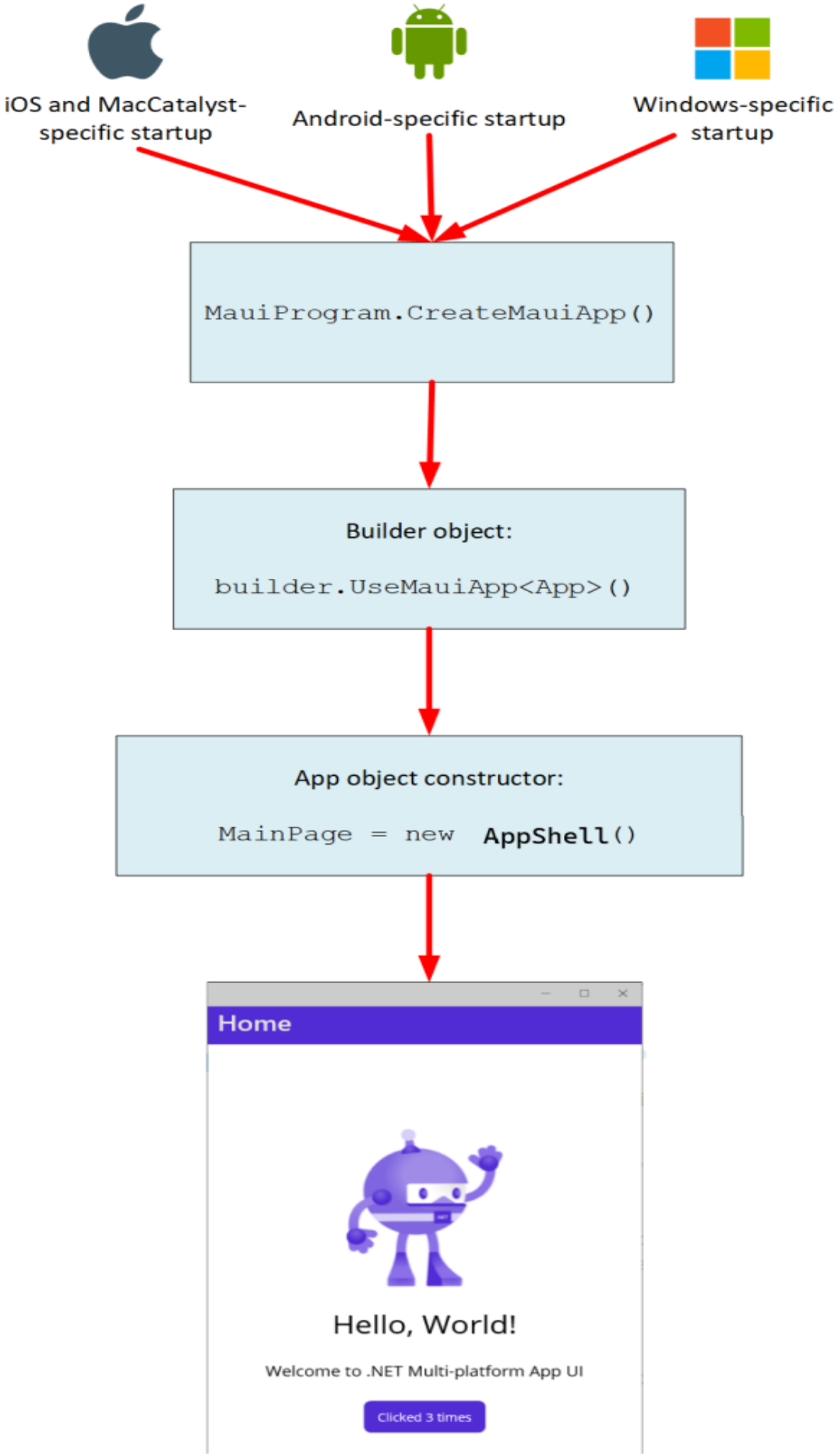
```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {

     fonts.AddFont("OpenSans-Regular.ttf","OpenSansRegular");
    fonts.AddFont("OpenSans-Semibold.ttf","OpenSansSemibold");

            });

        return builder.Build();
    }
}
```

Platforms يحتوي هذا المجلد على ملفات وموارد التعليمات البرمجية للتهيئة الخاصة بنظام التشغيل platform-specific initialization توجد مجلدات خاصة بأنظمة Android, iOS, MacCatalyst, Tizen, and Windows في وقت التشغيل، يبدأ تشغيل التطبيق بطريقة خاصة بالنظام الأساسي، تقوم مكتبات MAUI بتلخيص جزء كبير من عملية بدء التشغيل، ولكن ملفات التعليمات البرمجية في هذه المجلدات توفر آلية لربط عملية التهيئة المخصصة الخاصة بك، النقطة المهمة هي أنه عند اكتمال التهيئة، تستدعي التعليمات البرمجية الخاصة بالنظام، الأسلوب MauiProgram.CreateMauiApp method الذي يقوم بعد ذلك بإنشاء وتشغيل الكائن App object وتشغيله، كما هو موضح سابقاً.

على سبيل المثال، يحتوي ملف MainApplication.cs في مجلد Android وملف AppDelegate.cs في مجلد iOS وMacCatalyst وملف App.xaml.cs في مجلد Windows على التجاوزات:

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

توضح الصورة التالية تدفق التحكم عند بدء تشغيل تطبيق .NET MAUI.


iOS and MacCatalyst-specific startup
Android-specific startup
Windows-specific startup
MauiProgram.CreateMauiApp()
Builder object:
builder.UseMauiApp<App>()
App object constructor:
MainPage = new AppShell()
Home
Hello, World!
Welcome to .NET Multi-platform App UI
Clicked 3 times

Project resources موارد المشروع

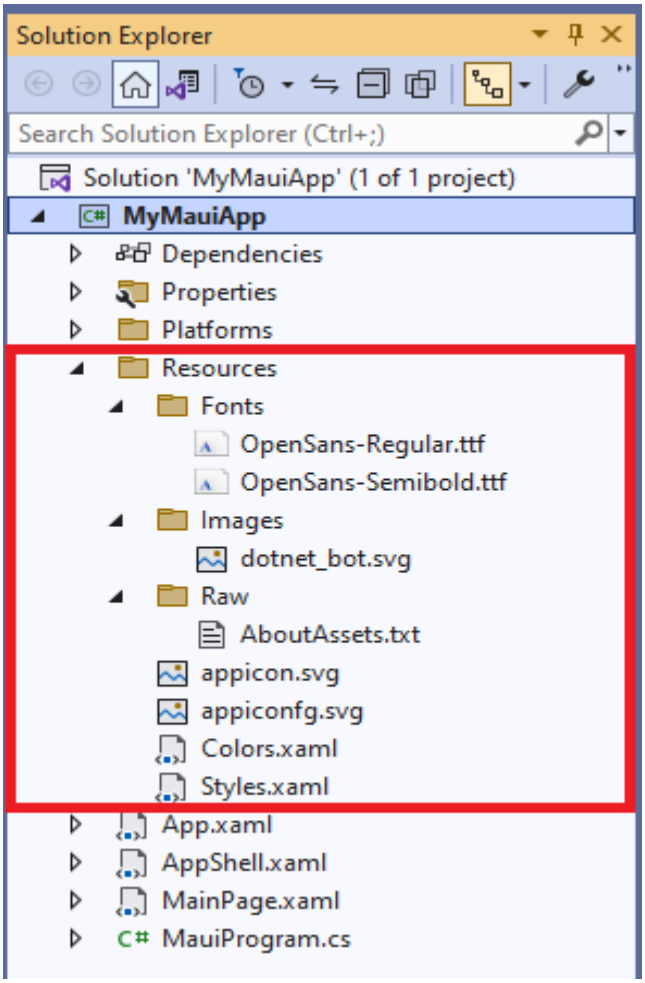
يتضمن ملف المشروع (.csproj) للمشروع الرئيسي عدة أقسام جديرة بالملاحظة، تحدد PropertyGroup الأولية أطر عمل النظام الذي يستهدفه المشروع، وعناصر مثل عنوان التطبيق، والمعرف، والإصدار، وإصدار العرض، وأنظمة التشغيل المدعومة title, ID, version, display version, and supported operating systems يمكنك تعديل هذه الخصائص حسب الضرورة.

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
   <TargetFrameworks>net6.0-android;net6.0-ios;net6.0-
maccatalyst </TargetFrameworks>
   <TargetFrameworks
Condition="$([MSBuild]::IsOSPlatform('windows'))">$(TargetFram
eworks);net6.0-windows10.0.19041.0</TargetFrameworks>

<!-- Uncomment to also build the tizen app. You will need to install
tizen by following this: https://github.com/Samsung/Tizen.NET -->
<!-- <TargetFrameworks>$(TargetFrameworks);net6.0-
tizen</TargetFrameworks> -->

   <OutputType>Exe</OutputType>
   <RootNamespace>MyMauiApp</RootNamespace>
   <UseMaui>true</UseMaui>
   <SingleProject>true</SingleProject>
   <ImplicitUsings>enable</ImplicitUsings>

   <!-- Display name -->
   <ApplicationTitle>MyMauiApp</ApplicationTitle>

   <!-- App Identifier -->
  <ApplicationId>com.companyname.mymauiapp</ApplicationId>
  <ApplicationIdGuid>272B9ECE-E038-4E53-8553-
E3C9EA05A5B2</ApplicationIdGuid>

  <!-- Versions -->
  <ApplicationDisplayVersion>1.0</ApplicationDisplayVersion>
  <ApplicationVersion>1</ApplicationVersion>

  <TargetPlatformMinVersion Condition=
"$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)'
)) == 'windows'">10.0.17763.0</TargetPlatformMinVersion>

  <SupportedOSPlatformVersion Condition=
"$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)'
)) == 'android'">21.0</SupportedOSPlatformVersion>

  </PropertyGroup>
    ...

</Project>
```

ItemGroup يتيح لك المقطع الموجود أسفل مجموعة الخصائص الأولية تحديد صورة ولون لشاشة البداية التي تظهر أثناء تحميل التطبيق، قبل ظهور النافذة الأولى، يمكنك أيضا تعيين المواقع الافتراضية للخطوط والصور والموارد التي يستخدمها التطبيق.

```
<Project Sdk="Microsoft.NET.Sdk">

    ...

  <ItemGroup>
     <!-- App Icon -->
     <MauiIcon Include="Resources\appicon.svg"
          ForegroundFile="Resources\appiconfg.svg"
          Color="#512BD4" />

   <!-- Splash Screen -->
   <MauiSplashScreen Include="Resources\appiconfg.svg"
         Color="#512BD4"
         BaseSize="128,128" />

   <!-- Images -->
   <MauiImage Include="Resources\Images\*" />
   <MauiImage Update="Resources\Images\dotnet_bot.svg"
            BaseSize="168,208" />

   <!-- Custom Fonts -->
   <MauiFont Include="Resources\Fonts\*" />

<!-- Raw Assets (also remove the "Resources\Raw" prefix) -->
 <MauiAsset Include="Resources\Raw\**"
   LogicalName="%(RecursiveDir)%(Filename)%(Extension)"/>
    </ItemGroup>

    ...

</Project>
```

في نافذة مستكشف الحلول Solution Explorer في Visual Studio يمكنك توسيع مجلد الموارد Resources لمشاهدة هذه العناصر، يمكنك إضافة أي خطوط وصور وموارد رسومية أخرى يتطلبها التطبيق إلى هذا المجلد والمجلدات الفرعية.



يجب عليك تسجيل أي خطوط تمت إضافتها إلى مجلد الخطوط باستخدام كائن منشئ التطبيق the app builder object عند بدء تشغيل التطبيق. تذكر أن الأسلوب CreateMauiApp في فئة MauiProgram يسجل الخطوط باستخدام الأسلوب ConfigureFonts

```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
             ...
            .ConfigureFonts(fonts =>
            {
     fonts.AddFont("OpenSans-Regular.ttf","OpenSansRegular");
             });

        ...
    }
}
```

في هذا المثال السابق، يربط الأسلوب AddFont الخط بالاسم OpenSansRegular يمكنك تحديد هذا الخط عند تنسيق العناصر في وصف XAML لصفحة أو في قاموس مورد التطبيق:

```
<Application ...">
  <Application.Resources>
      <ResourceDictionary>
        ...
        <Style TargetType="Button">
          ...
         <Setter Property="FontFamily" Value=
"OpenSansRegular"/>
          ...
        </Style>

    </ResourceDictionary>
  </Application.Resources>
</Application>
```

استخدم مجلدات الموارد الموجودة في مجلدات Android, and iOS folders ضمن مجلد الأنظمة للموارد الخاصة بنظامي Android and iOS

اختبار بسيط:

١- في أي أسلوب من عناصر التطبيق يجب إنشاء النافذة الأولية التي يعرضها التطبيق؟

٢- أين يمكنك تنفيذ المنطق لمعالج حدث لعنصر تحكم، مثل حدث النقر لزر؟

حل الاختبار:

١ الدالة الإنشائية The constructor

٢ يمكنك تنفيذ المنطق في ملف التعليمات البرمجية الخلفي لصفحة XAML التي تحتوي على عنصر التحكم.

٤ تمرين، إنشاء تطبيق .NET MAUI الأول

في هذا التمرين، تبدأ في إنشاء تطبيق .NET MAUI. واجهة مستخدم التطبيق متعدد الأنظمة Multi-platform Application User Interface لسلسلة متاجر البقالة، يمكنك استخدام القالب لإنشاء التطبيق الافتراضي وتشغيله على Windows وفي محاكي Android في تمرين لاحق، يمكنك تعديل هذا التطبيق لتخصيص واجهة المستخدم، وإضافة الوظائف المطلوبة لتطبيق سلسلة متاجر البقالة.

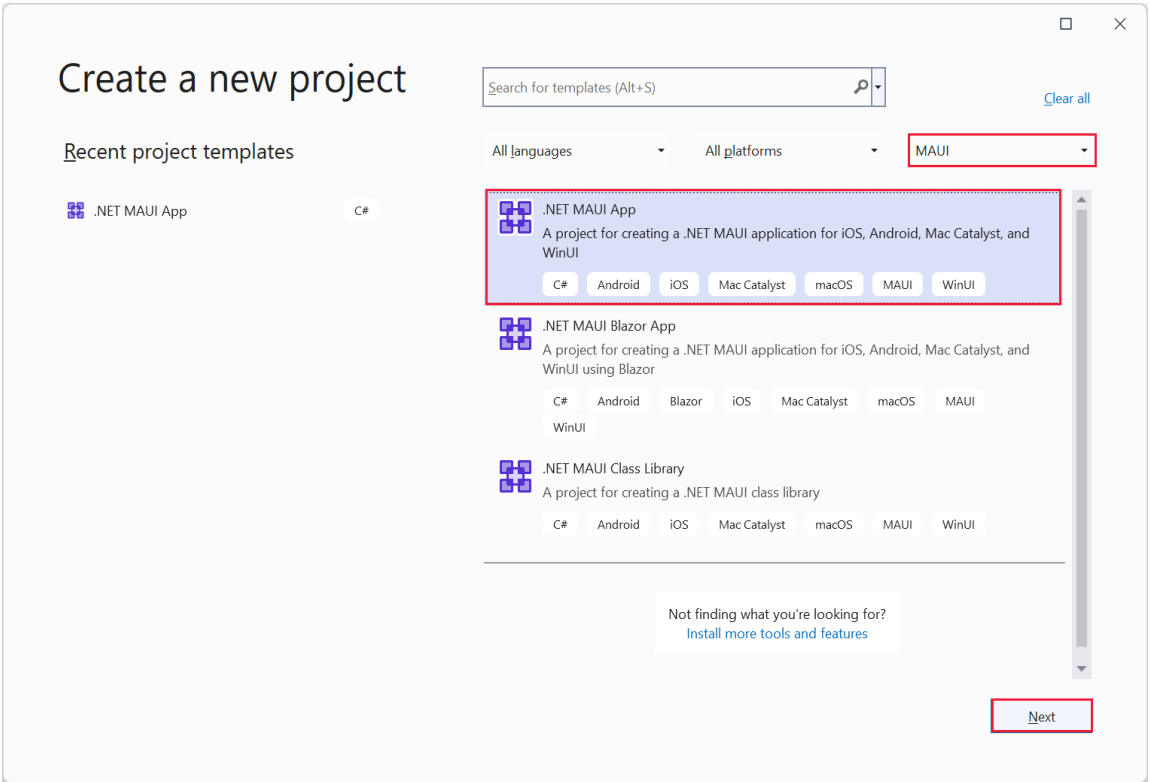
إنشاء مشروع جديد

١- افتح Visual Studio وقم بإنشاء حل جديد new solution يفتح هذا الإجراء معالج New Project

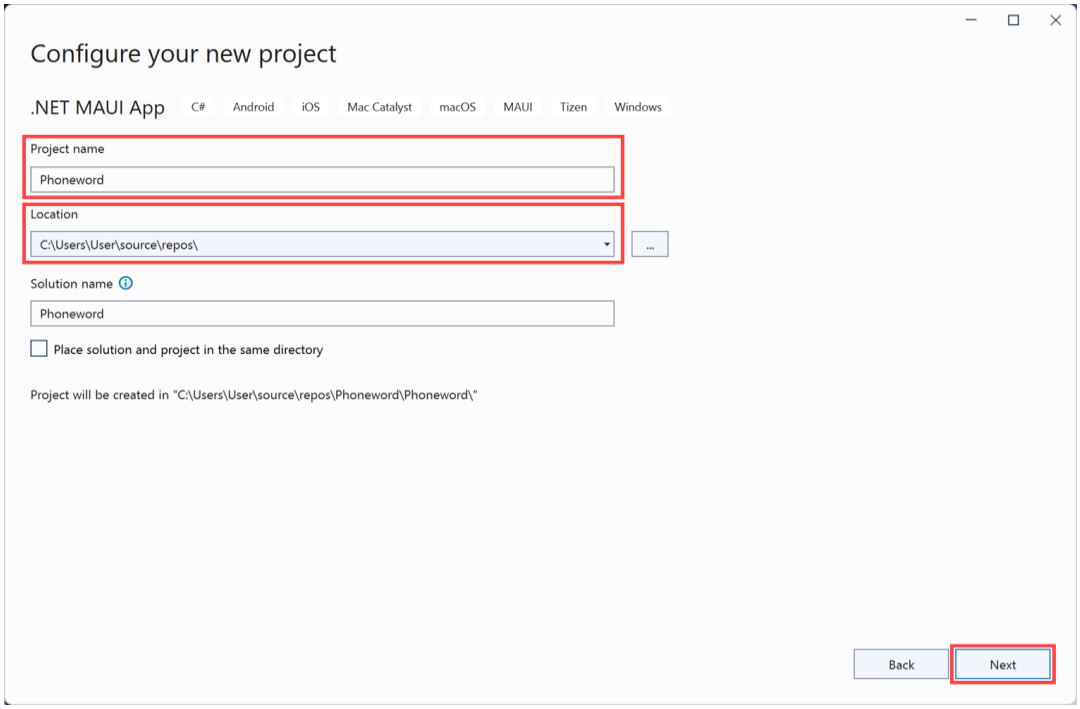
تلميح:

تأكد من تثبيت إطار عمل .NET MAUI. مع أحدث إصدار من Visual Studio 2022 v17.12 with .NET 9

٢- حدد نوع مشروع MAUI وحدد قالب .NET MAUI App. وحدد Next.



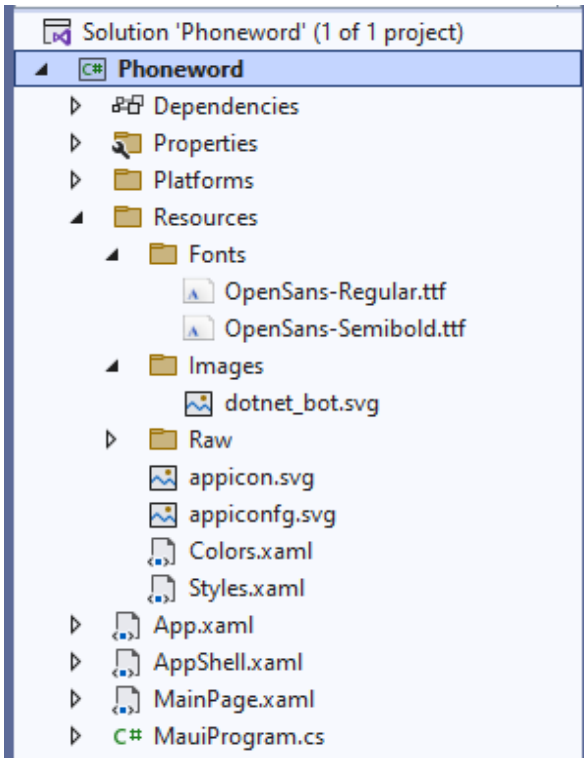
٣- في صفحة تكوين مشروعك الجديد، قم بتسمية المشروع Phoneword واحفظه في موقع من اختيارك. حدد التالي.



٤- حدد .NET 9.0 في القائمة المنسدلة Framework ثم حدد Create لإنشاء التطبيق.

فحص بنية الحل solution structure

١- في نافذة مستكشف الحلول، قم بتوسيع مشروع Phoneword قم بتوسيع مجلد الموارد Resources والمجلدات الفرعية الخاصة به، وقم بتوسيع عقدة App.xaml وعقدة AppShell.xaml وعقدة MainPage.xaml

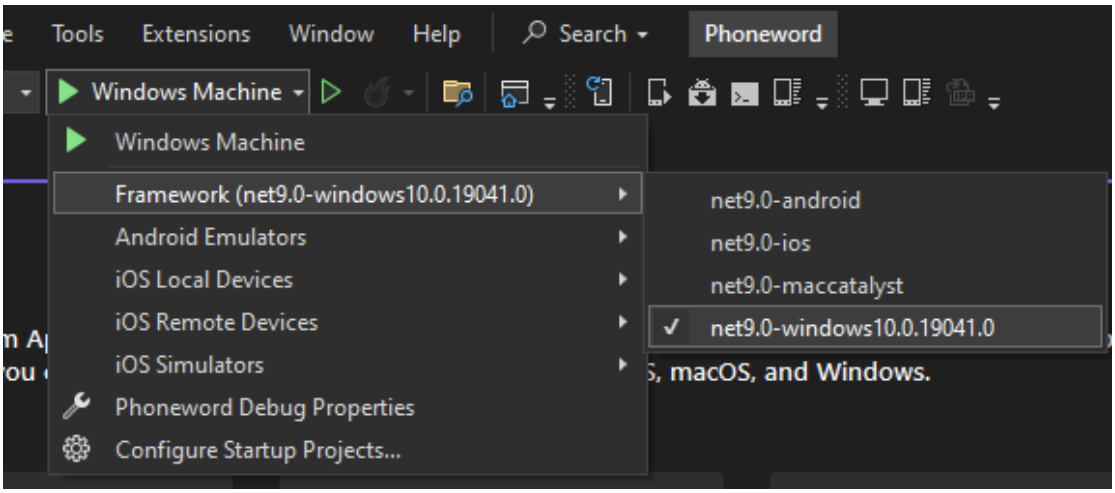


٢- في المشروع، لاحظ العناصر التالية:

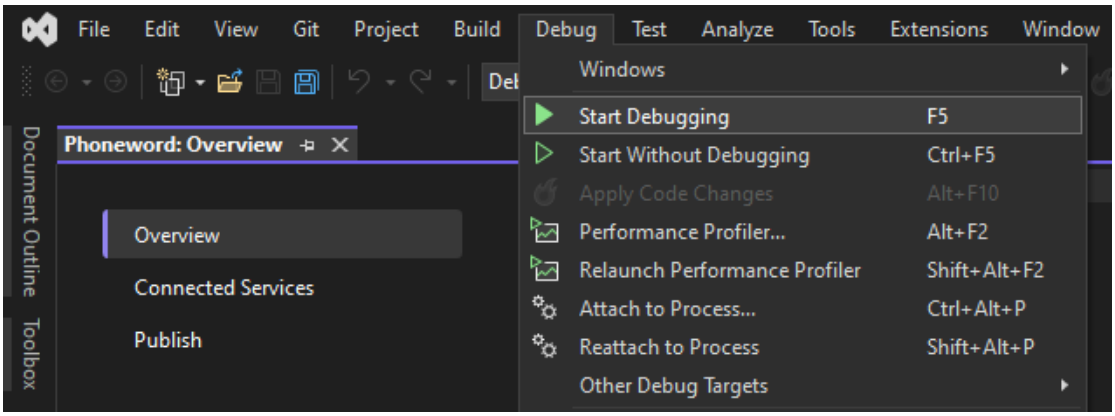
- يحتوي مجلد الموارد **Resources** على خطوط وصور وأصول مشتركة fonts, images, and assets تستخدمها جميع أنظمة التشغيل.
- يحتوي ملف **MauiProgram.cs** على التعليمات البرمجية التي تقوم بتكوين التطبيق ويحدد أنه يجب استخدام App class لتشغيل التطبيق.
- ينشئ ملف **App.xaml.cs** الدالة الإنشائية ل App class مثيلاً جديدًا لفئة **AppShell** class والذي يتم عرضه بعد ذلك في نافذة التطبيق.
- يحتوي ملف **AppShell.xaml** على التخطيط الرئيسي للتطبيق وصفحة البدء MainPage
- يحتوي الملف **MainPage.xaml** على تخطيط الصفحة layout for the page يتضمن هذا التخطيط التعليمة البرمجية XAML لزر button مع ملصق للتوضيح Click me وصورة تعرض ملف dotnet_bot.png هناك تسميتان أخريان أيضا.
- يحتوي ملف **MainPage.xaml.cs** على منطق التطبيق للصفحة application logic for the page على وجه التحديد، تتضمن فئة MainPage أسلوباً يسمى OnCounterClicked يتم تشغيله عندما يضغط المستخدم على الزر Click me button

إنشاء التطبيق وتشغيله على Windows

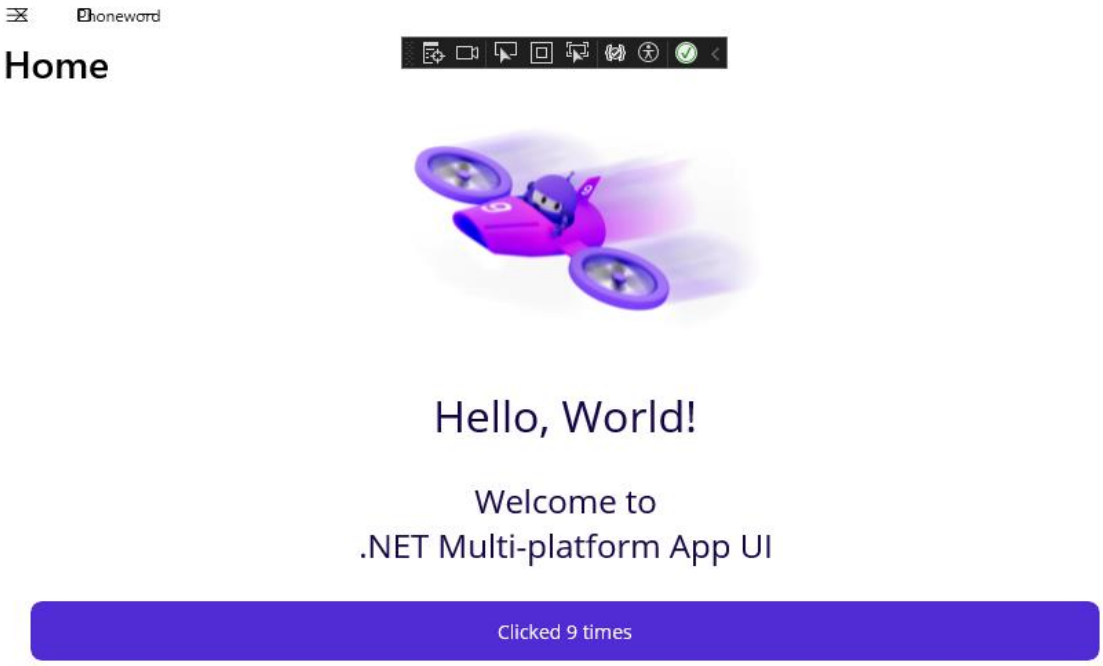
١- في شريط أدوات Visual Studio حدد ملف تعريف جهاز Windows Machine حدد net9.0-windows. من القائمة في مربع القائمة المنسدلة frameworks



٢- في القائمة Debug حدد Start Debugging يقوم هذا الإجراء بإنشاء التطبيق ونشره وتشغيله على Windows



٣- تحقق من بدء تشغيل إصدار Windows من التطبيق، اضغط الزر Click me button عدة مرات، يجب تحديث نص الزر، وزيادة العدد مع كل نقرة.



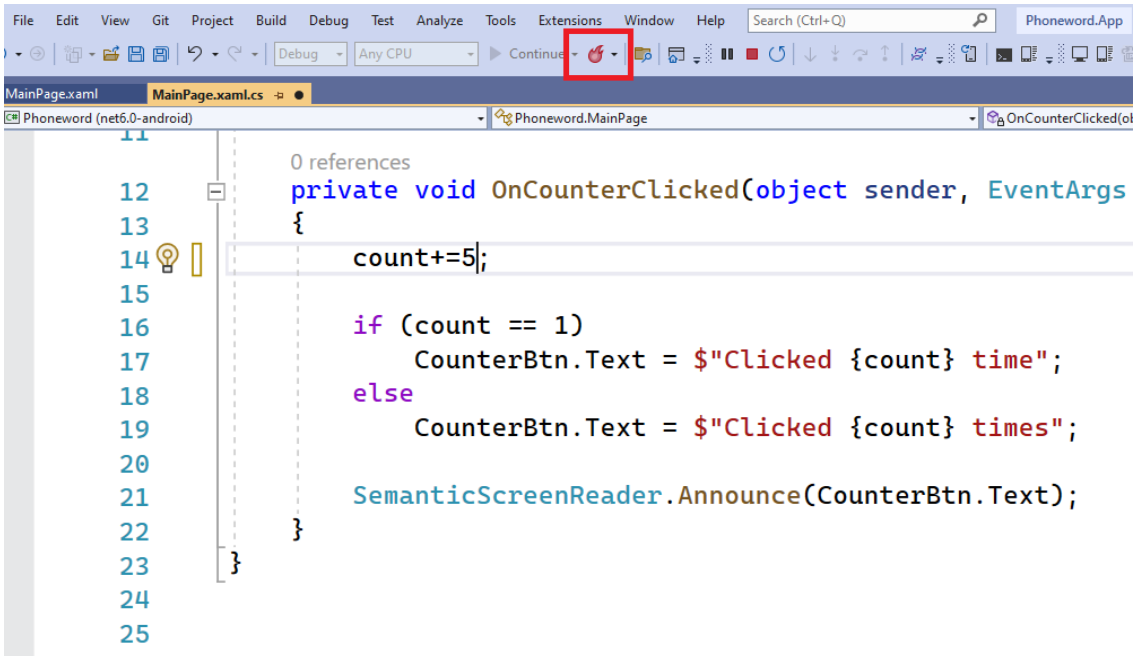
٤- في القائمة View حدد مستكشف الحلول Solution Explorer في نافذة مستكشف الحلول Solution Explorer قم بتوسيع عقدة MainPage.xaml وافتح ملف التعليمات البرمجية MainPage.xaml.cs في الخلفية.

٥- في الأسلوب OnCounterClicked method قم بتغيير العبارة التي تزيد من متغير count لإضافة 5 إلى هذا المتغير. بدلاً من ذلك:

```
private void OnCounterClicked(object sender, EventArgs e)
{
    count+=5; // update this
```

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

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

٦- في شريط أدوات Visual Studio حدد الزر Hot Reload اثناء تشغيل التطبيق.



٧- قم بالتبديل مرة أخرى إلى التطبيق وحدد الزر Click me تحقق من أن العدد يتزايد الآن بمقدار 5

ملحوظة:

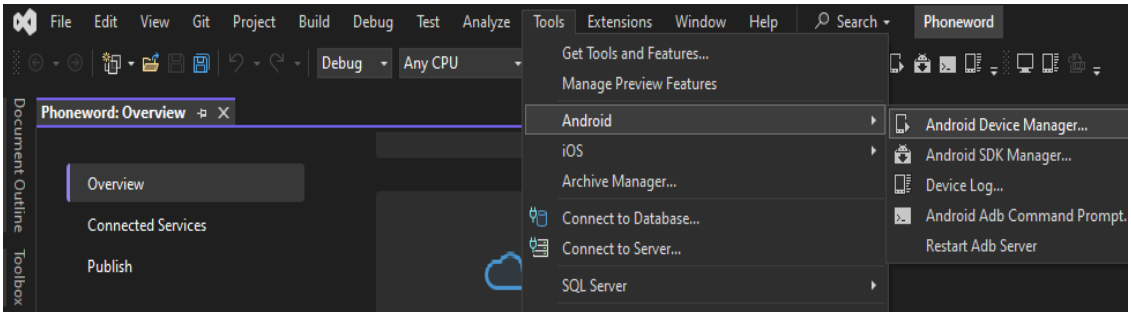
تتيح لك ميزة Hot Reload في Visual Studio تعديل التعليمات البرمجية أثناء تشغيل التطبيق في وضع التصحيح، لست بحاجة إلى إيقاف التطبيق لرؤية التغييرات، بالإضافة إلى تعديل التعليمات، يمكنك أيضًا إجراء تغييرات على ترميز XAML لأي صفحة، وستصبح هذه التغييرات مرئية في التطبيق قيد التشغيل.

٨- أغلق التطبيق وارجع إلى Visual Studio

إنشاء التطبيق وتشغيله على Android

١- في شريط أدوات Visual Studio حدد مشروع Phoneword

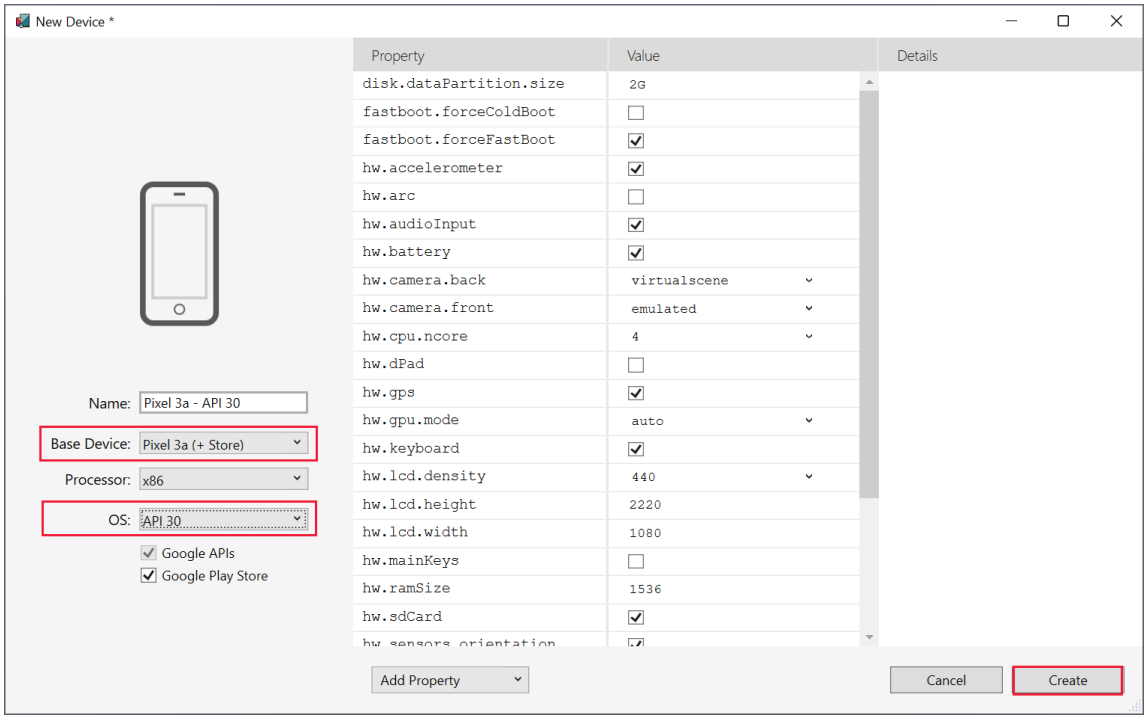
٢- في قائمة أدوات Tools حدد Android ثم حدد Android Device Manager
إذا تمت مطالبتك بالسماح لنظام التشغيل Android Device Manager بإجراء تغييرات على الكمبيوتر، فحدد Yes

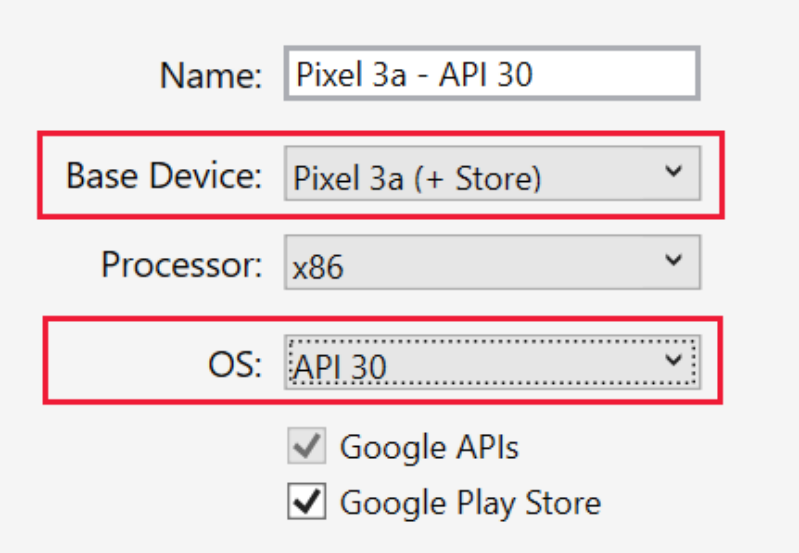


٣- في نافذة Android Device Manager حدد New + في نافذة New Device
حدد الجهاز الأساسي Pixel 3a (+ Store) base device وحدد نظام التشغيل API 30 OS وحدد Create انتظر حتى يتم تنزيل المكتبات المختلفة وتكوين الجهاز.

ملحوظة:

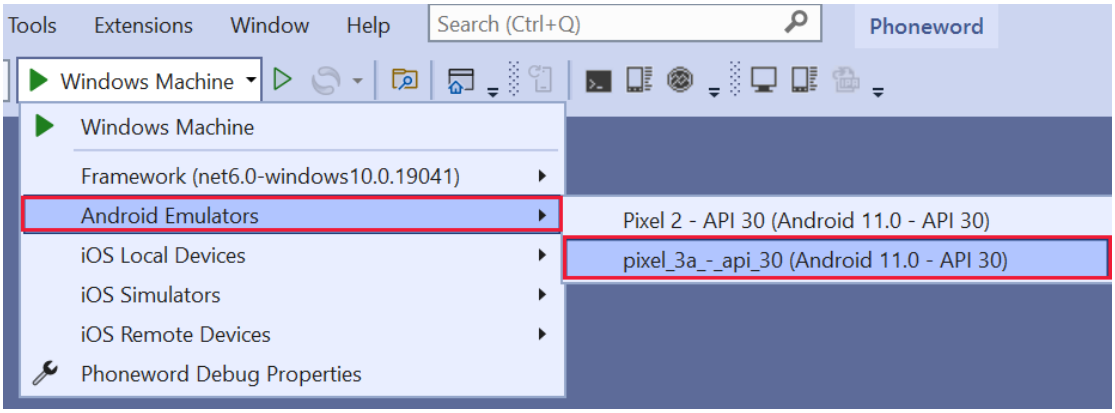
يستخدم المثال في هذا التمرين الجهاز الأساسي Pixel 3a (+ Store) ولكن يمكنك استخدام أجهزة أحدث. على سبيل المثال Pixel 5 - API 34



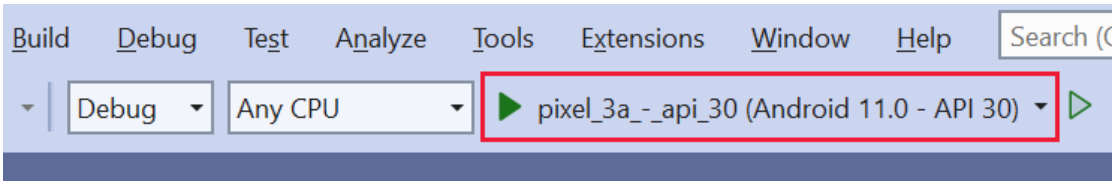


٤- عند إنشاء الجهاز، ارجع إلى Visual Studio

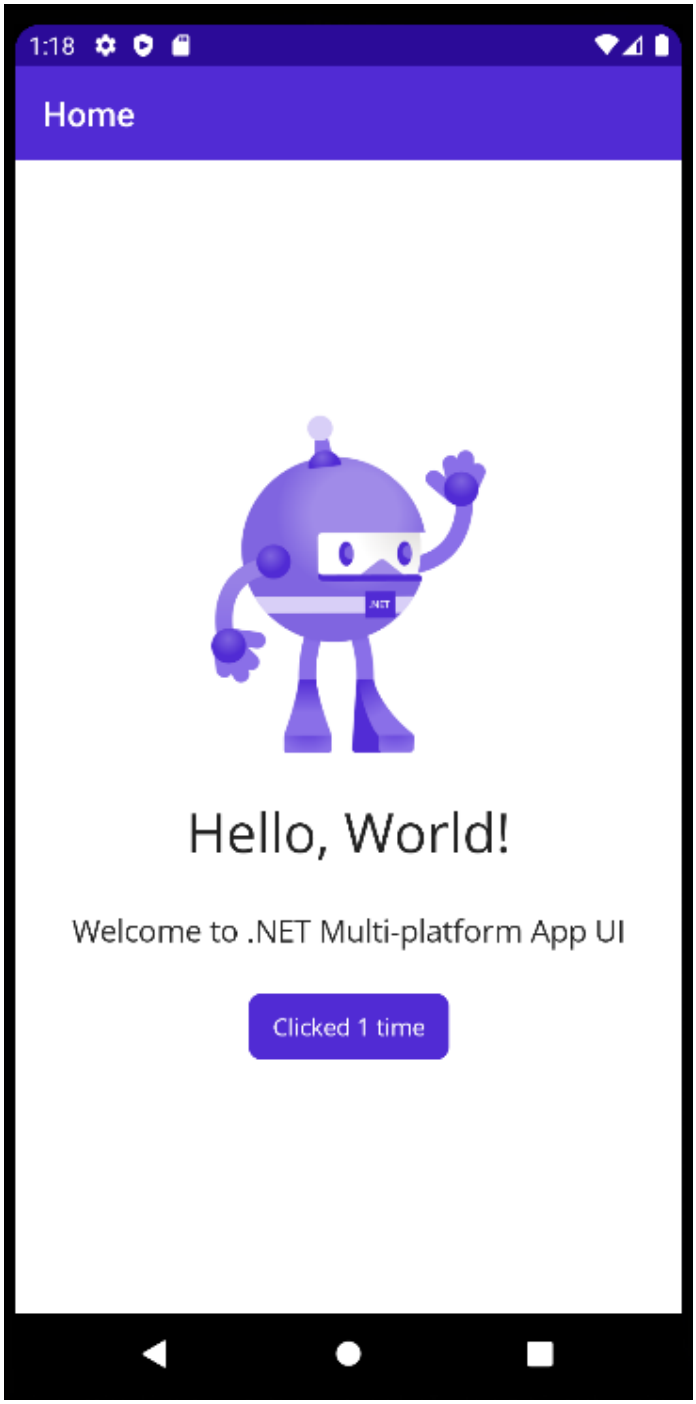
٥- في شريط أدوات Visual Studio في مربع القائمة المنسدلة تكوين التصحيح ة حدد ملف تعريف محاكيات Android حدد جهاز pixel_3a-api_30 الذي قمت بإنشائه.



٦- ابدأ تصحيح الأخطاء Start debugging باستخدام ملف تعريف pixel_3a-api_30 يقوم هذا الإجراء بإنشاء التطبيق ونشره وتشغيله على جهاز Android



٧- عند بدء تشغيل التطبيق على المحاكي (قد يستغرق هذا الإجراء عدة دقائق) حدد الزر Click me وتحقق من أن التطبيق يعمل بنفس الطريقة التي يعمل بها على Windows بالضبط.



٨- ارجع إلى Visual Studio وتوقف عن تصحيح الأخطاء.

٥ إضافة عناصر تحكم مرئية إلى تطبيق .NET MAUI

الآن بعد أن استخدمت قالب .NET MAUI. لإنشاء التطبيق، فإن الخطوة التالية هي إضافة واجهة المستخدم، وتنفيذ منطق واجهة المستخدم الأولي.

في هذا الدرس، يمكنك معرفة المزيد حول الكتل الإنشائية لتطبيق .NET MAUI. واجهة مستخدم التطبيق متعدد الأنظمة Multi-platform Application User Interface وهياكل التنقل navigation structures

ماذا يوجد في مشروع .NET MAUI.

للخلاصة، يحتوي مشروع .NET MAUI. في البداية على:

- ملف **MauiProgram.cs** الذي يحتوي على التعليمات البرمجية لإنشاء كائن التطبيق وتكوينه configuring the Application object
- ملفات **App.xaml** وملفات **App.xaml.cs** التي توفر موارد واجهة المستخدم UI resources وتنشئ النافذة الأولية للتطبيق.
- ملفات **AppShell.xaml** وملفات **AppShell.xaml.cs** التي تحدد الصفحة الأولية initial page للتطبيق، وتعالج تسجيل الصفحات لتوجيه التنقل بين الصفحات الأخرى.
- ملفات **MainPage.xaml** وملفات **MainPage.xaml.cs** التي تحدد التخطيط layout ومنطق واجهة المستخدم UI logic للصفحة المعروضة بشكل افتراضي في النافذة الأولية.

يمكنك إضافة صفحات أخرى pages إلى التطبيق حسب الضرورة، ويمكنك إنشاء المزيد من الفئات classes لتنفيذ منطق العمل الذي يتطلبه التطبيق.

يحتوي مشروع .NET MAUI. أيضاً على مجموعة افتراضية من موارد التطبيق مثل الصور والأيقونات والخطوط ورمز bootstrap الافتراضي لكل نظام أساسي.

فئة التطبيق The Application class

تمثل الفئة App class تطبيق .NET MAUI. ككل، يرث مجموعة افتراضية من السلوكيات من Microsoft.Maui.Controls.Application تذكر من الدرس السابق أن الفئة App class هي التي تقوم التعليمة البرمجية التمهيدية bootstrap code instantiates بإنشاء مثيلاً لها، وتحميلها لكل نظام تشغيل.

سيقوم منشئ App class constructor بدوره، عادةً بإنشاء مثيل لفئة AppShell class وتعيينه لخاصية MainPage property هذه التعليمة البرمجية هي التي تتحكم في الشاشة الأولى التي يراها المستخدم استنادًا إلى التعريفات في AppShell

تحتوي فئة التطبيق App class أيضا على:

* أساليب التعامل مع أحداث دورة حياة التطبيق، بما في ذلك وقت إرسال التطبيق إلى الخلفية (أي عندما يتوقف عن كونه التطبيق الموجود في الأمام foreground app)

* أساليب إنشاء نوافذ جديدة للتطبيق، يحتوي تطبيق .NET MAUI على نافذة واحدة بشكل افتراضي، ولكن يمكنك إنشاء المزيد من النوافذ وتشغيلها، وهو أمر مفيد في تطبيقات desktop and tablet applications

Shell

تقلل واجهة مستخدم التطبيق متعددة المنصات Shell (.NET MAUI) .NET. تعقيد تطوير التطبيقات، من خلال توفير الميزات الأساسية التي تتطلبها معظم التطبيقات، بما في ذلك:

* مكان واحد لوصف التسلسل الهرمي المرئي للتطبيق.

* تجربة مستخدم شائعة للتنقل.

* مخطط تنقل قائم على URI يسمح بالتنقل إلى أي صفحة في التطبيق.

* معالج بحث متكامل.

في تطبيق .NET MAUI Shell. يتم وصف التسلسل الهرمي المرئي للتطبيق في فئة فرعية من فئة Shell class يمكن أن تتكون هذه الفئة من ثلاثة عناصر هرمية رئيسية:

- `FlyoutItem` or `TabBar` يمثل FlyoutItem عنصرًا واحدًا أو أكثر في القائمة المنبثقة، ويجب استخدامه عندما يتطلب نمط التنقل للتطبيق وجود قائمة

منبثقة. يمثل TabBar شريط علامات التبويب السفلي، ويجب استخدامه عندما يبدأ نمط التنقل للتطبيق بعلامات تبويب سفلية ولا يتطلب نافذة منبثقة.

- Tab علامة التبويب، التي تمثل المحتوى المجمع، القابل للتنقل حسب علامات التبويب السفلية.
- ShellContent الذي يمثل كائنات ContentPage لكل علامة تبويب.

لا تمثل هذه الكائنات أي واجهة مستخدم، بل تمثل تنظيم التسلسل الهرمي المرئي للتطبيق. تأخذ Shell هذه الكائنات وتنتج واجهة مستخدم التنقل للمحتوى.

الصفحات Pages

الصفحات Pages هي أساس التسلسل الهرمي لواجهة المستخدم في .NET MAUI. داخل Shell حتى الآن، يتضمن الحل الذي رأيته فئة تسمى MainPage تشتق هذه الفئة من ContentPage وهو أبسط أنواع الصفحات page type وأكثرها شيوعًا، تعرض صفحة المحتوى content page محتوياتها ببساطة.

يحتوي .NET MAUI. أيضًا على العديد من أنواع الصفحات المضمنة الأخرى، بما في ذلك:

- `TabbedPage` هذه هي الصفحة الجذر المستخدمة للتنقل عبر علامة التبويب. تحتوي الصفحة المبوبة على كائنات صفحة فرعية تابعة؛ كائن واحد لكل علامة تبويب.
- `FlyoutPage` تمكنك من تنفيذ عرض تقديمي لنمط رئيسي/تفصيلي، تحتوي الصفحة المنبثقة على قائمة بالعناصر، عند تحديد عنصر، يظهر نمط عرض يعرض تفاصيل هذا العنصر.

تتوفر أنواع أخرى من الصفحات، وهي تُستخدم في الغالب لتمكين أنماط تنقل مختلفة في التطبيقات متعددة الشاشات.

أدوات العرض Views

تعرض صفحة المحتوى عادةً عرضًا، يتيح لك العرض استرداد البيانات، وتقديمها بنمط معين، العرض الافتراضي لصفحة المحتوى هو ContentView والذي يعرض العناصر كما هي، إذا قمت بتقليص العرض، فقد تختفي العناصر من العرض حتى تقوم بتغيير حجم العرض.

يتيح لك ScrollView عرض العناصر في نافذة تمرير؛ إذا قمت بتقليص النافذة، يمكنك التمرير لأعلى ولأسفل لعرض العناصر.

ال CarouselView نمط عرض قابل للتمرير، يتيح للمستخدم التمرير عبر مجموعة من العناصر.

يمكن CollectionView استرداد البيانات من مصدر بيانات مسمى، وتقديم كل عنصر باستخدام قالب كتنسيق. هناك العديد من أنواع أنماط العروض الأخرى المتاحة أيضًا.

عناصر التحكم والتخطيطات Controls and layouts

يمكن أن يحتوي نمط العرض على عنصر تحكم واحد مثل زر أو ملصق أو مربعات نص. (بالمعنى الدقيق للكلمة، فإن نمط العرض هو في حد ذاته عنصر تحكم، وبالتالي يمكن أن يحتوي نمط العرض على عرض آخر) ومع ذلك، فإن واجهة المستخدم المقيدة بعنصر تحكم واحد لن تكون مفيدة، لذا يتم وضع عناصر التحكم في تخطيط، يحدد التخطيط القواعد التي يتم بموجبها عرض عناصر التحكم بالنسبة لبعضها البعض، التخطيط هو أيضًا عنصر تحكم، لذا يمكنك إضافته إلى نمط عرض.

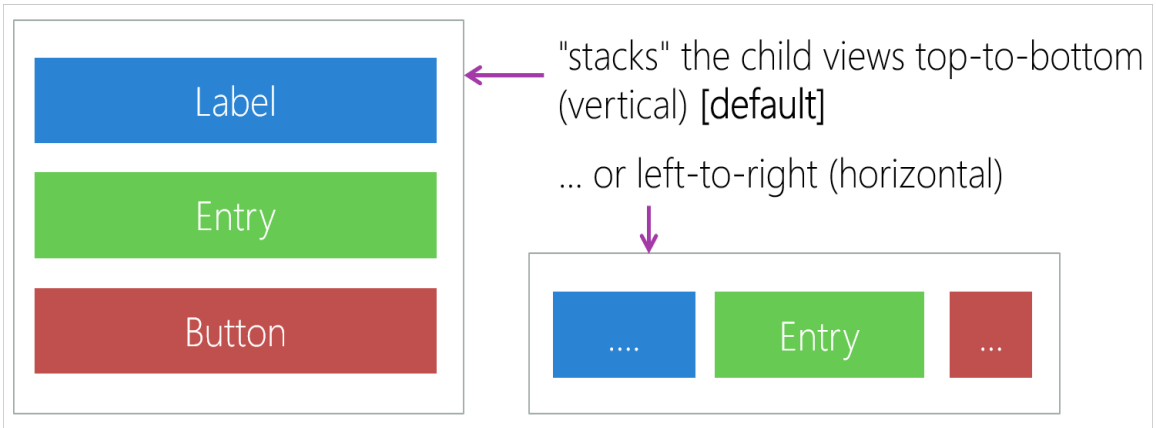
إذا نظرت إلى ملف MainPage.xaml الافتراضي، فسترى هذا التسلسل الهرمي للصفحة/العرض/التخطيط/عنصر التحكم، قيد التنفيذ. في التعليمات XAML هذه، يعد عنصر VerticalStackLayout مجرد عنصر تحكم آخر يسمح لك بضبط تخطيط عناصر التحكم الأخرى.

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

بعض عناصر التحكم controls الشائعة المستخدمة لتعريف التخطيطات layouts هي:

- VerticalStackLayout and HorizontalStackLayout وهما تخطيطان محسّنان للمكدسات، يعرضان عناصر التحكم في مجموعة من أعلى إلى أسفل أو من اليسار إلى اليمين، يتوفر أيضاً StackLayout والذي يحتوي على خاصية تسمى StackOrientation التي يمكنك تعيينها إلى Horizontal أو Vertical

أفقي أو عمودي على جهاز لوحي أو هاتف، يمكنك تعديل هذه الخاصية في التعليمات البرمجية للتطبيق، ضبط العرض إذا قام المستخدم بتدوير الجهاز:

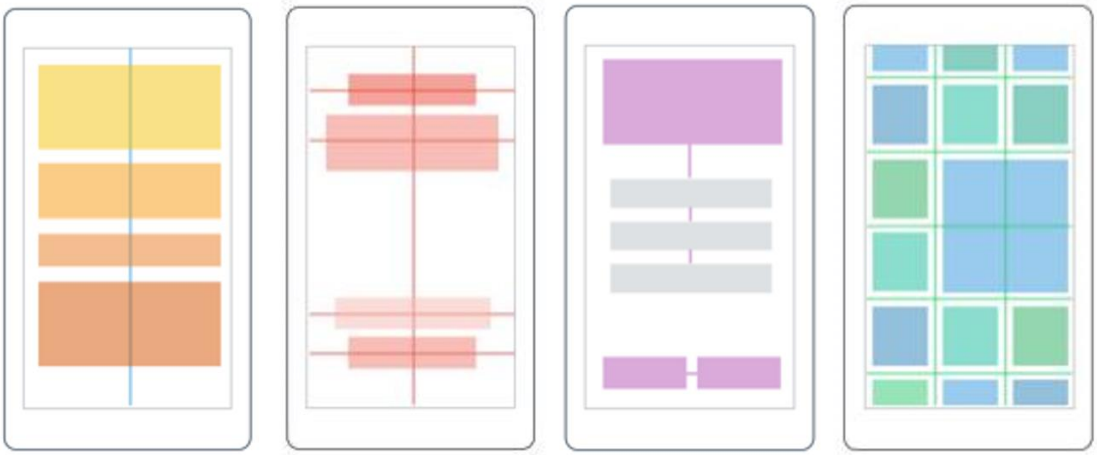


- AbsoluteLayout يسمح لك بتعيين إحداثيات دقيقة لعناصر التحكم.
- FlexLayout وهو مشابه لـ StackLayout باستثناء أنه يمكنك من التفاف عناصر التحكم التابعة التي يحتوي عليها، إذا لم يتم احتواؤها في صف أو عمود واحد، داخل الصفحة. يوفر هذا التخطيط أيضا خيارات للمحاذاة والتكيف مع أحجام الشاشات المختلفة، على سبيل المثال، يمكن لعنصر FlexLayout محاذاة عنصر التحكم الفرعي الخاص به إلى اليسار أو اليمين أو المنتصف عند ترتيبه عموديًا. عند المحاذاة أفقياً، يمكنك ضبط عناصر التحكم لضمان التباعد المتساوي.

يمكن استخدام FlexLayout أفقي داخل ScrollView لعرض سلسلة من الإطارات القابلة للتمرير أفقيًا (يمكن أن يكون كل إطار بحد ذاته FlexLayout مرتبًا عموديًا):

- Grid يحدد عناصر التحكم التابعة له وفقاً لموقع العمود والصف column, row الذي قمت بتحديده، يمكنك تحديد أحجام الأعمدة والصفوف ومداها، كما تريد، وبالتالي فإن تخطيطات الشبكة Grid لا تبدو بالضرورة وكأنها "رقعة شطرنج"

تلخص الصورة التالية السمات الرئيسية لهذه الأنواع الشائعة من التخطيطات layout:



تحتوي جميع عناصر التحكم على خصائص properties يمكنك تعيين القيم الأولية لهذه الخصائص باستخدام XAML في كثير من الحالات، يمكنك تعديل هذه الخصائص في التعليمات البرمجية C# الخاصة بتطبيقك. على سبيل المثال، تبدو التعليمات البرمجية التي تعالج حدث Clicked لزر Click me الافتراضي كما يلي:

```
int count = 0;
private void OnCounterClicked(object sender, EventArgs e)
{
    count+=5;

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

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

يعدل هذا الكود خاصية Text لعنصر التحكم CounterBtn في الصفحة، يمكنك حتى إنشاء صفحات وعناصر عرض وتخطيطات كاملة في التعليمات؛ **لست مضطرًا**

لاستخدام XAML على سبيل المثال، ضع في اعتبارك تعريف XAML التالي لصفحة:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Phoneword.MainPage">

    <ScrollView>
        <VerticalStackLayout>
            <Label Text="Current count: 0"
                Grid.Row="0"
                FontSize="18"
                FontAttributes="Bold"
                x:Name="CounterLabel"
                HorizontalOptions="Center" />

            <Button Text="Click me"
                Grid.Row="1"
                Clicked="OnCounterClicked"
                HorizontalOptions="Center" />
        </VerticalStackLayout>
    </ScrollView>
</ContentPage>
```

التعليمات البرمجية المكافئة له في C# كما يلي:

```
public partial class TestPage : ContentPage
{
    int count = 0;

   // Named Label - declared as a member of the class
    Label counterLabel;

    public TestPage()
    {
       var myScrollView = new ScrollView();

       var myStackLayout = new VerticalStackLayout();
       myScrollView.Content = myStackLayout;

       counterLabel = new Label
       {
           Text = "Current count: 0",
           FontSize = 18,
```

```
            FontAttributes = FontAttributes.Bold,
            HorizontalOptions = LayoutOptions.Center
        };
        myStackLayout.Children.Add(counterLabel);

        var myButton = new Button
        {
            Text = "Click me",
            HorizontalOptions = LayoutOptions.Center
        };
        myStackLayout.Children.Add(myButton);

        myButton.Clicked += OnCounterClicked;

        this.Content = myScrollView;
    }

 private void OnCounterClicked(object sender, EventArgs e)
    {
       count++;
       counterLabel.Text = $"Current count: {count}";

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

تعتبر تعليمات C# أكثر تفصيلاً، ولكن XAML توفر مرونة أكبر تسمح لك بتكييف واجهة المستخدم بشكل ديناميكي.

لعرض هذه الصفحة عند بدء تشغيل التطبيق، قم بتعيين class TestPage في AppShell باعتبارها ShellContent الرئيسية:

```
<ShellContent
    Title="Home"
    ContentTemplate="{DataTemplate local:TestPage}"
    Route="TestPage" />
```

ضبط التخطيط Tuning a layout

من المفيد إضافة مساحة صغيرة للتنفس حول عنصر التحكم control يحتوي كل عنصر تحكم على خاصية Margin "الهامش أو التباعد من الخارج" تحترمها التخطيطات، يمكنك التفكير في Margin على أنه عنصر تحكم يدفع العناصر الأخرى بعيدًا.

تحتوي جميع التخطيطات أيضًا على خاصية Padding "الهامش أو الحشو من الداخل" التي تمنع أيًا من عناصر التحكم التابعة لها من الاقتراب من حدود التخطيط، إحدى الطرق للتفكير في هذا المفهوم هي أن جميع عناصر التحكم موجودة في مربع، وهذا المربع له جدران مبطنة.

هناك إعداد مفيد آخر للمسافة البيضاء وهو خاصية Spacing في VerticalStackLayout أو HorizontalStackLayout تحدد هذه الخاصية المسافة بين كافة العناصر الفرعية التابعة للتخطيط children of the layout القيمة مضافة إلى Margin الهامش الخاص بعنصر التحكم نفسه، لذا فإن المسافة البيضاء الفعلية هي الهامش بالإضافة إلى المسافة.

اختبار بسيط:

١- أي من هذه الأنواع ليس نوع تخطيط layout type في NET MAUI.

DockableLayout
AbsoluteLayout
StackLayout
Grid

٢- ما الفئة class المستخدمة لإنشاء شاشة create a screen في NET MAUI.

حل الاختبار:

١- DockableLayout ليس نوع تخطيط

٢- Page هي الفئة الأساسية base class لـ ContentPage وأنواع الصفحات الأخرى

٦ تمرين، إنشاء تطبيق مترجم رقم الهاتف

في هذا التمرين، يمكنك إنشاء واجهة مستخدم لتطبيق الاتصال الهاتفي، وتنفيذ المنطق وراء واجهة المستخدم هذه.

يمكنك إنشاء واجهة مستخدم تستفيد من إمكانيات واجهة المستخدم لـ .NET MAUI. وحزمة .NET MAUI Essentials للاتصال الهاتفي.

يتيح التطبيق للمستخدم كتابة نص في حقل إدخال، ويترجم هذا النص إلى أرقام رقمية. ويستخدم الحروف التي تظهر على لوحة مفاتيح الهاتف كأساس للترجمة. على سبيل المثال، تُترجم الحروف cab إلى ٢٢٢ لأن الرقم ٢ يحتوي على الحروف الثلاثة a وb وc.

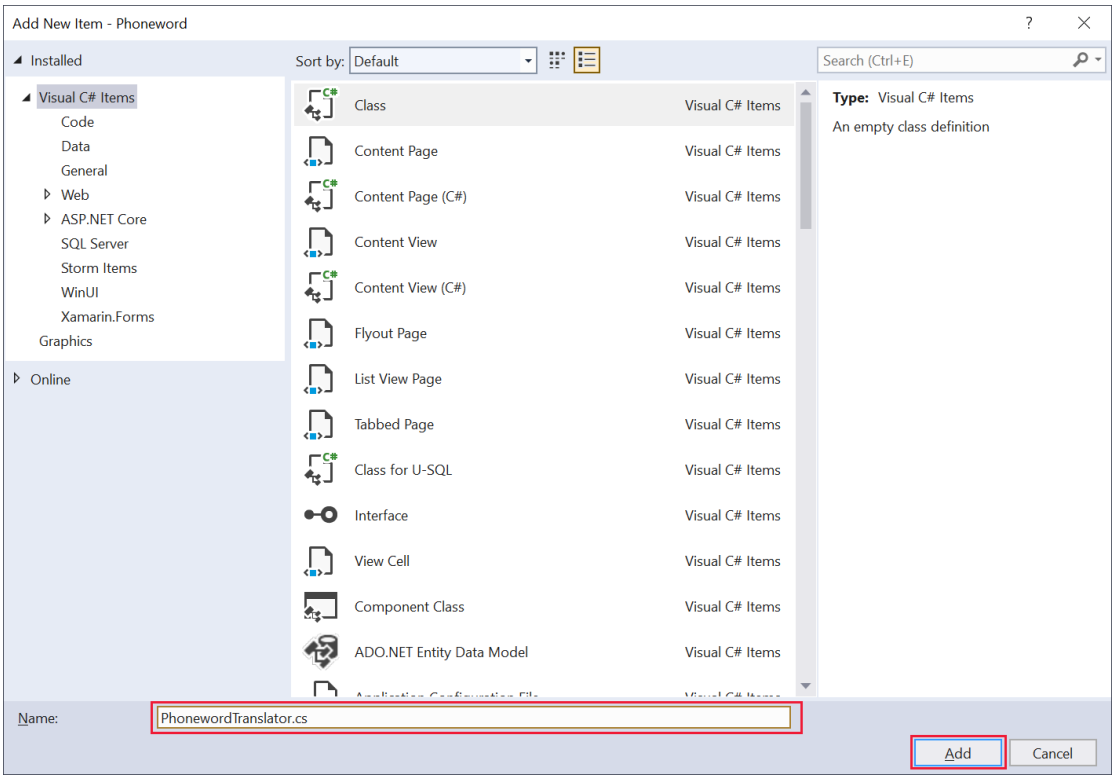
يمكنك الاستمرار في حل Phoneword الذي أنشأته في التمرين السابق.

إضافة ملف C# جديد إلى التطبيق new C# source file

١- افتح حل Phoneword في Visual Studio إذا لم يكن مفتوحاً بالفعل.

٢- في نافذة مستكشف الحلول Solution Explorer انقر بزر الماوس الأيمن فوق مشروع Phoneword وحدد Add ثم حدد Class

٣- في مربع الحوار إضافة عنصر جديد Add New Item قم بتسمية ملف Class الفئة PhonewordTranslator.cs ثم حدد إضافة Add



إضافة منطق الترجمة Add the translation logic

استبدل محتوى ملف class بالتعليمات البرمجية التالية واحفظ الملف، الأسلوب الثابت static method ToNumber في PhonewordTranslator class يترجم الرقم من نص أبجدي رقمي إلى رقم هاتف رقمي عادي.

```
using System.Text;
namespace Core;

public static class PhonewordTranslator
{
    public static string ToNumber(string raw)
    {
        if (string.IsNullOrWhiteSpace(raw))
            return null;
        raw = raw.ToUpperInvariant();

        var newNumber = new StringBuilder();
        foreach (var c in raw)
        {
            if (" -0123456789".Contains(c))
                newNumber.Append(c);
            else
            {
                var result = TranslateToNumber(c);
                if (result != null)
                    newNumber.Append(result);
                // Bad character?
                else
                    return null;
            }
        }
        return newNumber.ToString();
    }

  static bool Contains(this string keyString, char c)
    {
        return keyString.IndexOf(c) >= 0;
    }
```

```
    static readonly string[] digits = {
    "ABC", "DEF", "GHI", "JKL", "MNO", "PQRS", "TUV",
"WXYZ"
    };

    static int? TranslateToNumber(char c)
    {
        for (int i = 0; i < digits.Length; i++)
        {
            if (digits[i].Contains(c))
                return 2 + i;
        }
        return null;
    }
}
```

إنشاء واجهة المستخدم Create the UI

١- افتح الملف MainPage.xaml في مشروع Phoneword

٢- قم بإزالة عنصر التحكم ScrollView ومحتوياته، مع ترك عنصر التحكم ContentPage فقط:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Phoneword.MainPage">

</ContentPage>
```

٣- أضف عنصر التحكم `VerticalStackLayout` الاتجاه الرأسي، بمسافة spacing of 15 وبحشو padding of 20 إلى ContentPage:

```
   <VerticalStackLayout Spacing="15" Padding="20">

   </VerticalStackLayout>
```

٤- أضف ملصق تسمية `Label` إلى عنصر التخطيط VerticalStackLayout

```
    <VerticalStackLayout ...>
        <Label Text = "Enter a Phoneword"
               FontSize ="20"/>
    </VerticalStackLayout>
```

٥- أضف عنصر `Entry` إلى VerticalStackLayout أسفل `Label` يوفر عنصر التحكم `Entry` مربع نص، يتاح للمستخدم إدخال البيانات داخله، في هذه التعليمات، تمنح الخاصية x:Name عنصر التحكم اسماً. يمكن الرجوع إلى عنصر التحكم هذا في التعليمات الخاصة بالتطبيق لاحقاً:

```
    <Entry x:Name = "PhoneNumberText"
           Text = "1-555-NETMAUI" />
```

٦- أضف عنصري `Button` إلى VerticalStackLayout بعد عنصر `Entry` كلا الزرين حالياً لا يفعلان شيئا، والثاني معطل في البداية.

يمكنك إضافة التعليمات البرمجية لمعالجة الحدث `Clicked` لهذين الزرين في المهمة التالية:

```
    <Button x:Name = "TranslateButton"
            Text = "Translate"
            Clicked = "OnTranslate"/>
    <Button x:Name = "CallButton"
            Text = "Call"
            IsEnabled = "False"
            Clicked = "OnCall"/>
```

حدث الاستجابة للزر TranslateButton

١- في نافذة مستكشف الحلول Solution Explorer قم بتوسيع MainPage.xaml وافتح ملف التعليمات البرمجية MainPage.xaml.cs في الخلف.

٢- في فئة `MainPage class` قم بإزالة متغير `count` وأسلوب `OnCounterClicked` يجب أن تبدو الفئة `class` على النحو التالي:

```
namespace Phoneword;

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
}
```

٣- أضف متغير `string variable translatedNumber` والأسلوب التالي `OnTranslate` إلى MainPage Class بعد الدالة الإنشائية، يسترد الأسلوب OnTranslate رقم الهاتف من خاصية Text في عنصر Entry ويمرره للأسلوب `static ToNumber` لفئة `PhonewordTranslator class` التي قمت بإنشائها سابقاً.

```
public partial class MainPage : ContentPage
{
    ...
 string? translatedNumber;

 private void OnTranslate(object sender, EventArgs e)
  {
    string enteredNumber = PhoneNumberText.Text;
    translatedNumber =
PhonewordTranslator.ToNumber(enteredNumber);

     if (!string.IsNullOrEmpty(translatedNumber))
     {
         // TODO:
     }
     else
     {
         // TODO:
     }
 }
}
```

ملحوظة: يمكنك ملء أجزاء TODO المفقودة من هذه التعليمات البرمجية في الخطوة التالية، أيضاً النقاط ... تدل على التعليمات البرمجية الأخرى.

٤- في الأسلوب OnTranslate أضف تعليمة برمجية لتغيير خاصية Text لزر الاتصال، لإضافة رقم الهاتف الذي تمت ترجمته بنجاح، يمكنك استخدام القيمة التي قمت بتخزينها في الحقل translatedNumber قم أيضًا بتمكين الزر وتعطيله بناءً على الترجمة الناجحة. على سبيل المثال، إذا أرجع TranslateNumber قيمة خالية null فقم بتعطيل الزر، ولكن إذا نجح، فقم بتمكينه.

```
private void OnTranslate(object sender, EventArgs e)
{
    string enteredNumber = PhoneNumberText.Text;
```

```
    translatedNumber =
Core.PhonewordTranslator.ToNumber(enteredNumber);

    if (!string.IsNullOrEmpty(translatedNumber))
    {
        CallButton.IsEnabled = true;
        CallButton.Text = "Call " + translatedNumber;
    }
    else
    {
        CallButton.IsEnabled = false;
        CallButton.Text = "Call";
    }
}
```

إنشاء أسلوب الحدث the event method لزر CallButton

١- أضف أسلوب معالجة الحدث OnCall في نهاية فئة MainPage class يستخدم هذه الأسلوب العمليات غير المتزامنة asynchronous operations لذا قم بتمييزها على أنها غير متزامنة `async`

```
public partial class MainPage : ContentPage
{

    ...
async void OnCall(object sender, System.EventArgs e)
    {

    }
}
```

٢- في الأسلوب OnCall اطلب من المستخدم، باستخدام الأسلوب `Page.DisplayAlert` السؤال عما إذا كان يرغب في الاتصال بالرقم.

المعلمات الخاصة بـ DisplayAlert هي عنوانا ورسالة، وسلسلتان نصيتان a title, a message, and two strings تستخدمان لنص الزرين Accept and Cancel قبول وإلغاء، وهي تعيد قيمة منطقية returns a Boolean تشير إلى ما إذا كان الزر Accept قد تم الضغط عليه لإغلاق مربع الحوار.

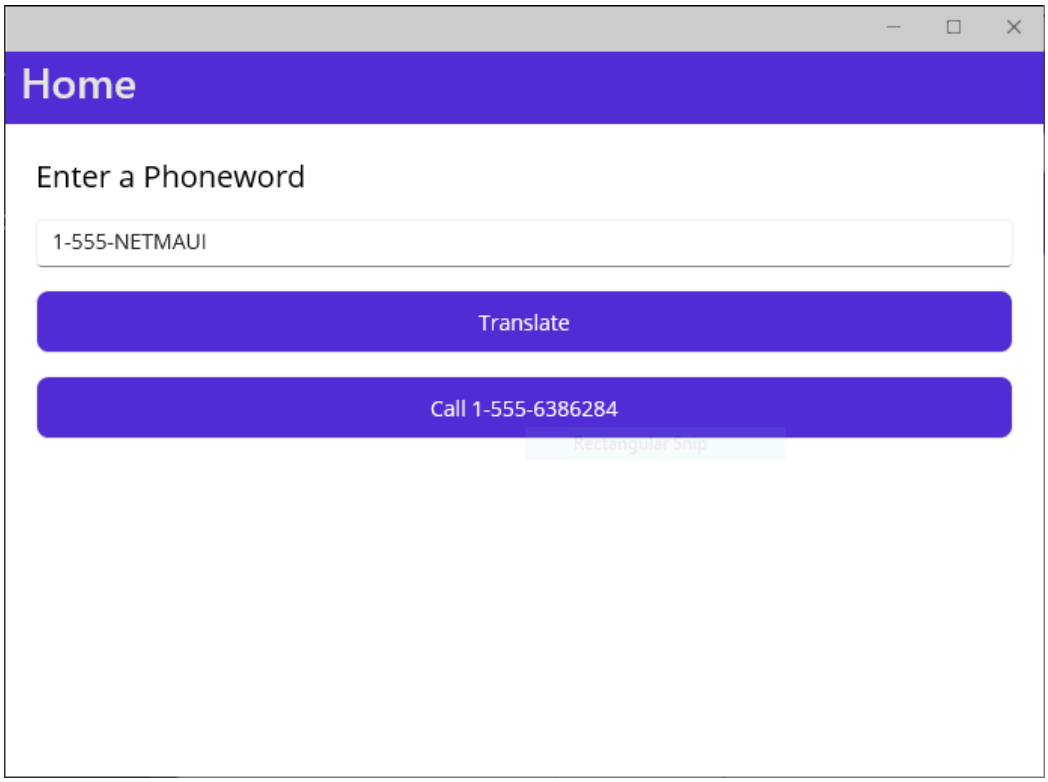
```
async void OnCall(object sender, System.EventArgs e)
{
```

```
    if (await this.DisplayAlert(
        "Dial a Number",
        "Would you like to call " +
translatedNumber + "?",
        "Yes",
        "No"))
    {
        // TODO: dial the phone
    }
}
```

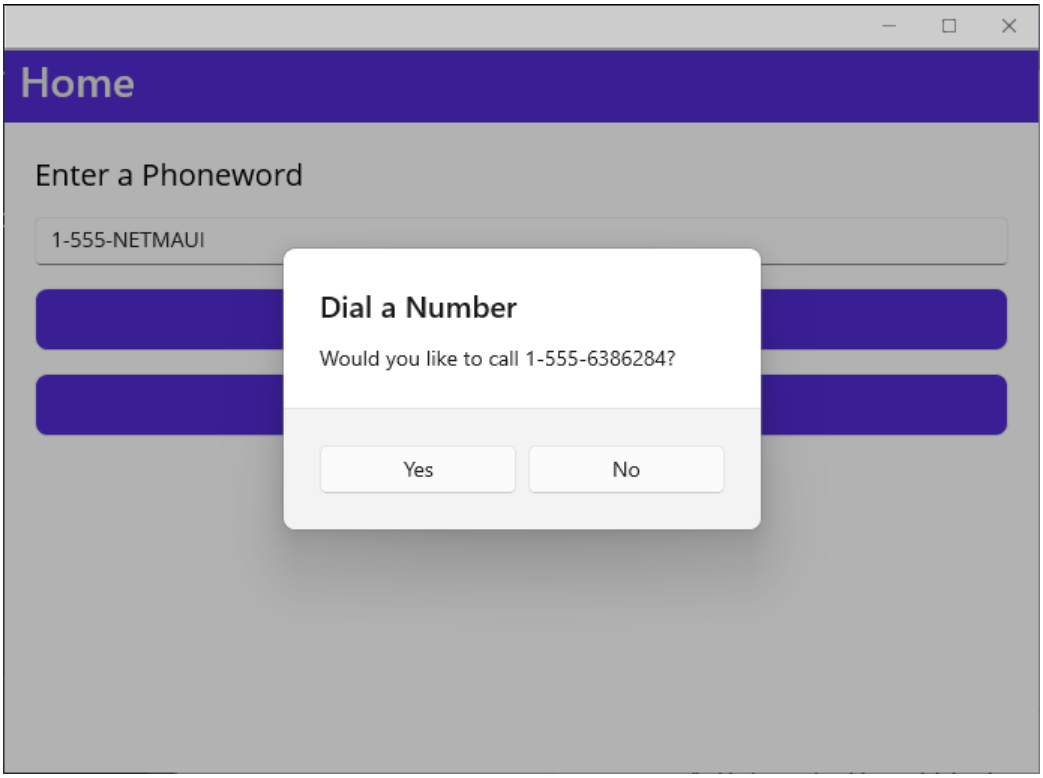
اختبر التطبيق

١- في شريط أدوات Visual Studio حدد جهاز Windows وابدأ تصحيح الأخطاء.

٢- اضغط على الزر Translate لتحويل النص الافتراضي إلى رقم هاتف صالح. يجب تغيير التسمية التوضيحية على زر Call إلى Call 1-555-6386284



٣- اضغط على زر Call تأكد من ظهور رسالة تطلب منك تأكيد العملية. حدد No



٤- ارجع إلى Visual Studio وتوقف عن تصحيح الأخطاء.

طلب رقم الهاتف Dial the phone number

١- في ملف التعليمات البرمجية MainPage.xaml.cs الخلفي، قم بتحرير الأسلوب OnCall واستبدل تعليق TODO بالكتل التالية try/catch

```
async void OnCall(object sender, System.EventArgs e)
{
 if (await this.DisplayAlert(
   "Dial a Number",
  "Would you like to call " + translatedNumber + "?",
   "Yes",
   "No"))
 {
   try
 {
  if (PhoneDialer.Default.IsSupported &&
!string.IsNullOrWhiteSpace(translatedNumber))

PhoneDialer.Default.Open(translatedNumber);
  }
  catch (ArgumentNullException)
```

```
 {
  await DisplayAlert("Unable to dial", "Phone number
was not valid.", "OK");
  }
  catch (Exception)
  {
   // Other error has occurred.
  await DisplayAlert("Unable to dial", "Phone dialing
failed.", "OK");
        }
    }
}
```

توفر الفئة PhoneDialer class في مساحة اسم Microsoft.Maui.ApplicationModel.Communication namespace تجريدًا لوظيفة طلب الهاتف (وغيرها) لأنظمة Windows, Android, iOS and iPadOS, and macOS platforms يحاول أسلوب Open static استخدام برنامج الاتصال الهاتفي لاتصال الرقم المقدم كمعلمة.

توضح الخطوات التالية كيفية تحديث بيان تطبيق Android لتمكين Android من استخدام برنامج الاتصال الهاتفي، تتبع تطبيقات Windows, iOS, MacCatalyst نفس المبدأ العام، باستثناء تحديد إمكانية مختلفة في البيان اعتمادا على نظام التشغيل.

٢- في نافذة مستكشف الحلول Solution Explorer قم بتوسيع مجلد Platforms وقم بتوسيع مجلد Android وانقر بزر الماوس الأيمن فوق ملف AndroidManifest.xml وحدد Open with > Automatic Editor واختر Selector (XML) OK

٣- أضف مقتطف XML التالي داخل عقدة البيان manifest node بعد المحتوى الموجود لهذه العقدة.

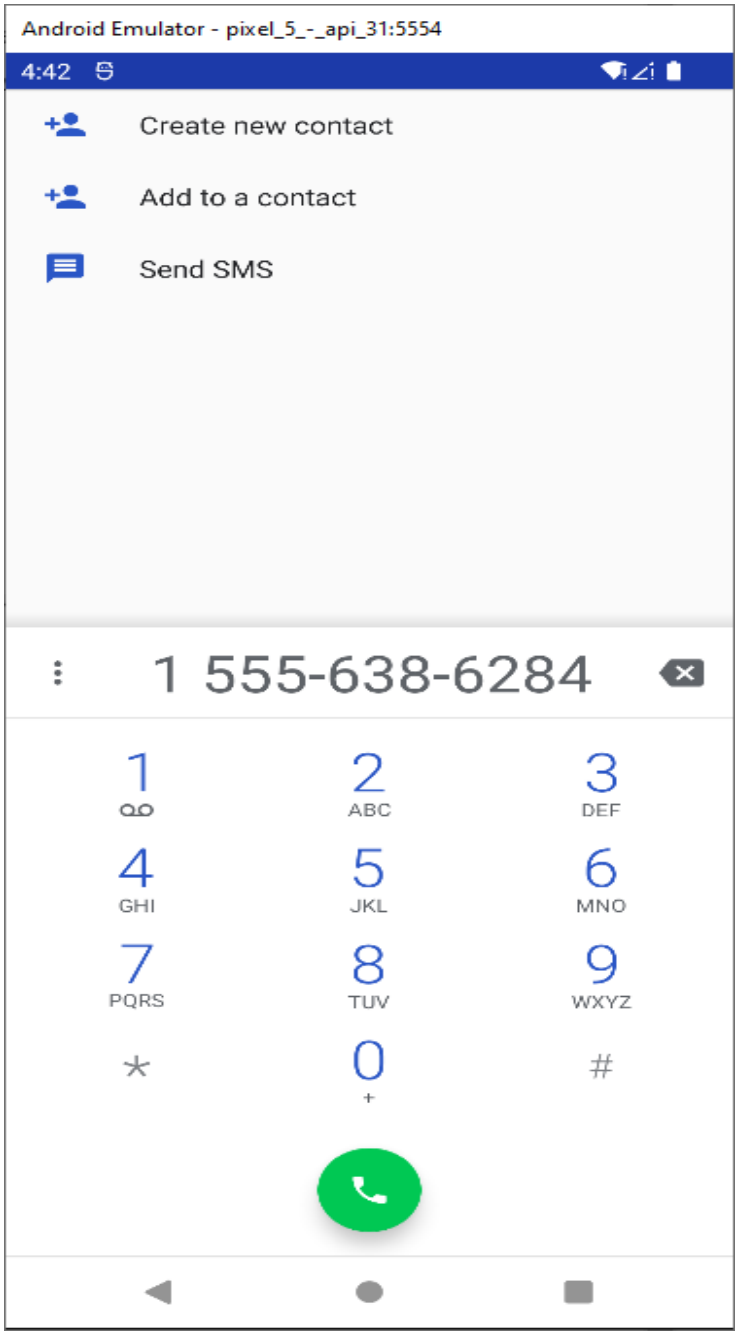
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  ...
  <queries>
     <intent>
    <action android:name="android.intent.action.DIAL" />
        <data android:scheme="tel"/>
     </intent>
  </queries>
</manifest>
```

٤- احفظ الملف

٥- في شريط أدوات Visual Studio حدد جهاز Android Emulators (أو ما شابه) وابدأ تصحيح الأخطاء.

٦- عندما يظهر التطبيق في المحاكي (قد يستغرق الأمر بضع دقائق) أدخل رقم هاتف (أو اقبل الافتراضي) حدد Translate ثم حدد Call

٧- في تنبيه الاتصال برقم Dial a Number حدد نعم Yes تحقق من ظهور برنامج الاتصال الهاتفي Android مع الرقم الذي قدمته في التطبيق.



٨- ارجع إلى Visual Studio وتوقف عن تصحيح الأخطاء.

التلخيص

في هذا التمرين، أضفت واجهة مستخدم custom UI مخصصة إلى تطبيقك، باستخدام الصفحات وطرق العرض pages and views أضفت أيضاً دعماً لإجراء مكالمة باستخدام واجهات برمجة التطبيقات الخاصة بنظام التشغيل platform-specific APIs المتوفرة في Android

٧ الملخص

يوفر .NET MAUI طريقة لإنشاء تطبيقات عبر الأنظمة cross-platform apps التي يمكن تشغيلها على أجهزة Windows, Android, macOS, and iOS devices

في هذه الوحدة، ستتعلم:

- البنية الأساسية لواجهة مستخدم التطبيق متعدد الأنظمة .NET MAUI.
- إنشاء تطبيق .NET MAUI.
- تعريف واجهة مستخدم مشتركة للأنظمة المدعومة من .NET MAUI.
- نشر تطبيق .NET MAUI من Visual Studio.
- استخدم .NET MAUI للوصول إلى واجهات برمجة التطبيقات الخاصة بنظام التشغيل.

لقد أنشأت تطبيق .NET MAUI يعمل على Windows, Android كما استخدمت واجهات برمجة تطبيقات .NET MAUI للوصول إلى برنامج الاتصال الهاتفي على كلا النظامين. يعمل هذا التطبيق كدليل على صحة المفهوم المحدد في المقدمة، على وجه التحديد، شرعت في إنشاء تطبيق:

- يعمل على Windows, Android والأجهزة المدعومة الأخرى.
- تقليل من وقت التطوير.
- يمكنه الوصول إلى برنامج الاتصال الهاتفي.

الوحدة الثانية

إنشاء واجهة مستخدم في تطبيق .NET MAUI باستخدام XAML

التعرف على كيفية تصميم واجهة مستخدم لتطبيق .NET MAUI باستخدام XAML

الأهداف التعليمية:

بنهاية هذه الوحدة، ستتمكن من وصف:

- فوائد استخدام XAML بدلاً من تعريف واجهة المستخدم لتطبيق .NET MAUI في C#
- الأنواع المتاحة لتعريف تطبيق .NET MAUI باستخدام XAML والخصائص التي تكشفها هذه الأنواع.
- كيفية إنشاء الصفحات وعناصر التحكم وتعيين خصائصها باستخدام XAML
- كيفية التعامل مع أحداث واجهة المستخدم handle UI events وربطها في XAML
- كيفية إنشاء واستخدام ملحقات ترميز XAML markup extensions
- كيفية تعيين قيم خاصة بنظام التشغيل في علامات XAML

محتويات الوحدة:

١- مقدمة

٢- ميزات استخدام XAML

٣- الأنواع والخصائص في .NET MAUI XAML.

٤- معالجة الحدث Event handling في XAML

٥- تمرين: إنشاء أول صفحة XAML

٦- ملحقات علامات أو ترميز XAML markup extensions

٧- قيم خاصة بنظام التشغيل Platform-specific values في XAML

٨- تمرين: إضافة سلوك إلى صفحة XAML

٩- الملخص

١ المقدمة

يتيح لك .NET MAUI. إنشاء واجهة مستخدم التطبيق ديناميكياً، باستخدام تعليمات C# ومع ذلك، في بعض الأحيان يكون من المناسب والأكثر كفاءة تعريف واجهة المستخدم بشكل ثابت، توفر Extensible Application Markup Language (XAML) طريقة لتخطيط واجهة المستخدم في وقت التجميع أو التحويل البرمجي. يوفر وصف XAML لواجهة المستخدم أيضاً درجة من التوثيق، ما يسمح لك بفهم الطريقة التي يتم بها تقديم الواجهة للمستخدم بسرعة دون الحاجة إلى التعمق في التعليمات البرمجية للتطبيق.

افترض أنك تعمل في شركة المرافق الكهربائية كمطور أجهزة الهواتف، أنت مسؤول عن تحسين تطبيق الهواتف للشركة الذي يستخدمه المهندسون عند زيارة مباني العملاء، حالياً، يتم إنشاء واجهة مستخدم التطبيق باستخدام التعليمات البرمجية C# ومع ذلك، بدأت تلاحظ أنه أصبح من الصعب إدارة تحديثات واجهة المستخدم، ترجع هذه الصعوبة إلى أن التطبيق أصبح أكثر تعقيدًا، أصبح من الصعب فهم منطق السلوك الأساسي، لأنه مختلط مع التعليمات البرمجية لواجهة المستخدم.

تريد إيجاد حلّ يقدم فصلًا واضحًا بين واجهة المستخدم والسلوك، يتيح فصل واجهة المستخدم والسلوك لخبير التصميم التركيز على ما يفعله بشكل أفضل، وسيتوفر لديك الوقت للتركيز على ترميز سلوك التطبيق، يتيح لك .NET MAUI. تحديد واجهة المستخدم الخاصة بك باستخدام XAML تمنحك XAML فصلًا واضحًا بين واجهة المستخدم UI والسلوك، كما تسهل XAML استخدام خبير التصميم وأدوات التصميم.

في هذه الوحدة، ستتعلم كيفية إنشاء تطبيق .NET MAUI. يحدد صفحاته وعناصر التحكم الخاصة به باستخدام XAML بدلاً من تعليمات C# يتيح لك إنشاء واجهة المستخدم على XAML فصل جميع التعليمات البرمجية لواجهة المستخدم عن تعليمات السلوك لتسهيل إدارة كليهما.

٢ ميزات استخدام XAML

XAML هي لغة ترميز يمكنك استخدامها لإنشاء واجهة المستخدم الخاصة بك بدلاً من تعليمات C# باستخدام XAML يمكنك تقسيم تعليمات واجهة المستخدم والسلوك لتسهيل إدارتهما.

في هذا الدرس، ستقارن استخدام XAML مع تعريف تخطيط واجهة المستخدم باستخدام تعليمات C# ستتعلم أيضًا بعض فوائد استخدام XAML كلغة ترميز لتحديد واجهة المستخدم الخاصة بك.

ما هي لغة التمييز؟ markup language

لغة الترميز هي لغة كمبيوتر يمكنك استخدامها لتقديم عناصر متعددة في مستند، يمكن وصف العناصر باستخدام علامات معرّفة مسبقًا. العلامات لها معاني محددة في سياق المجال الذي يتم فيه استخدام المستند.

على سبيل المثال، يمكنك استخدام لغة ترميز النص التشعبي (HTML) لإنشاء صفحة ويب، وعرضها في متصفح الويب، لست بحاجة إلى فهم جميع العلامات التي نستخدمها في المثال التالي؛ ما هو مهم أن ترى هو أن هذه التعليمة البرمجية تصف مستندًا يحتوي على النص "Hello World!" كمحتوى له.

```
<!DOCTYPE html>
<html>
    <body>
        <p>Hello <b>World</b>!</p>
    </body>
</html>
```

من المحتمل أنك عملت بالفعل باستخدام لغة ترميز، ربما قمت بإنشاء صفحة ويب باستخدام HTML أو ربما قمت بتعديل تعريفات Extensible Markup Language (XML) في ملف Visual Studio project.csproj تقوم أدوات الإنشاء من Microsoft بتحليل هذا الملف ومعالجته.

من الشائع معالجة الملفات التي تحتوي على لغة الترميز وتفسيرها بواسطة أدوات البرامج الأخرى، هذه الطبيعة التفسيرية للغة الترميز تبين بالضبط كيف يُقصد بالعمل باستخدام XAML ومع ذلك، تساعد أدوات البرمجيات التي تفسرها في إنشاء واجهة مستخدم التطبيق.

ما هي XAML؟

XAML هي لغة ترميز تعريفية تم إنشاؤها بواسطة Microsoft تم تصميم XAML لتبسيط عملية إنشاء واجهة المستخدم في التطبيقات.

تحتوي مستندات XAML التي تقوم بإنشائها على عناصر تصف عناصر واجهة مستخدم التطبيق بشكل تعريفي، ضع في اعتبارك أن هذه العناصر في XAML تمثل مباشرة إنشاء الكائنات، بمجرد تعريف عنصر في XAML يمكنك الوصول إليه في ملفات التعليمات البرمجية الخلفية، وتحديد السلوك باستخدام لغة C#

الفرق بين .NET MAUI XAML and Microsoft XAML

يستند XAML إلى مواصفات Microsoft 2009 XAML ومع ذلك، تحدد هذه المواصفات قواعد بناء جملة اللغة فقط syntax كما هو الحال مع Windows Presentation Foundation (WPF), Universal Windows Platform (UWP), and WinUI 3 وكلها تستخدم XAML فإن العناصر التي تعلنها في XAML سوف تتغير.

ظهرت لغة XAML لأول مرة في عام 2006 مع WPF إذا كنت تعمل مع Microsoft XAML لفترة من الوقت، فيجب أن تبدو قواعد XAML مألوفة.

هناك بعض الاختلافات الرئيسية بين .NET MAUI flavor of XAML. ولغة XAML المستخدمة من قبل أدوات واجهة المستخدم الأخرى، البنية والمفاهيم متشابهة، ولكن بعض أسماء الفئات والخصائص مختلفة.

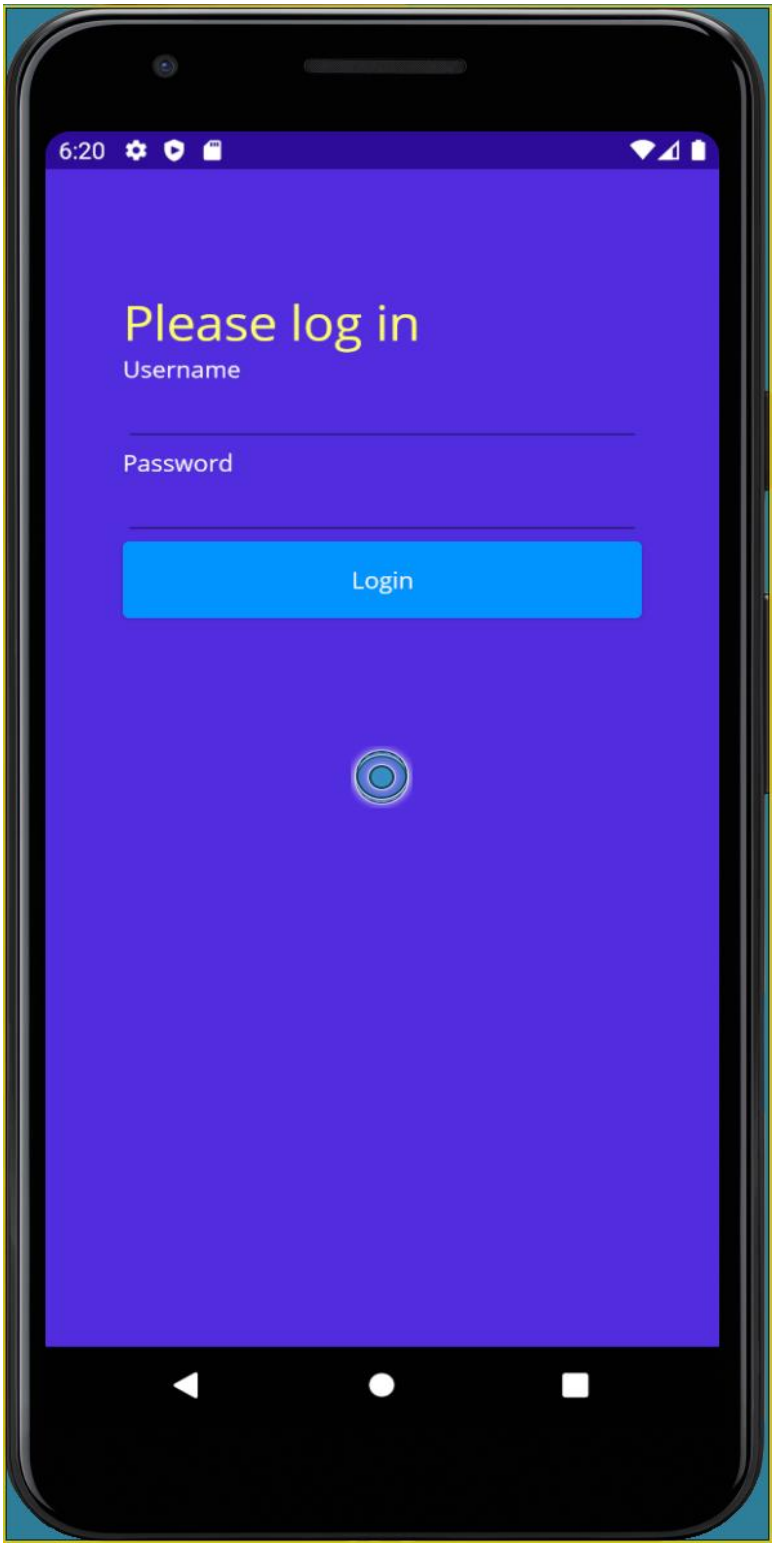
إنشاء واجهة مستخدم باستخدام .NET MAUI XAML UI by using

أفضل طريقة لرؤية XAML قيد التنفيذ هي إلقاء نظرة على مثال لنوع صفحة ContentPage مكتوب بتعليمات موجودة بلغة C# يمكنك بعد ذلك مقارنتها بصفحة أخرى لها نفس واجهة المستخدم المعرفة باستخدام XAML

لنفترض أن لديك التعليمات البرمجية `ContentPage` التالية في تطبيقك:

```
namespace MauiCode;

public partial class MainPage : ContentPage
{
    Button loginButton;
    VerticalStackLayout layout;
    public MainPage()
    {
        this.BackgroundColor = Color.FromArgb("512bdf");

        layout = new VerticalStackLayout
        {
            Margin = new Thickness(15, 15, 15, 15),
            Padding = new Thickness(30, 60, 30, 30),
            Children =
            {
             new Label { Text = "Please log in", FontSize =
30, TextColor = Color.FromRgb(255, 255, 100) },
              new Label { Text = "Username", TextColor =
Color.FromRgb(255, 255, 255) },
              new Entry (),
              new Label { Text = "Password", TextColor =
Color.FromRgb(255, 255, 255) },
              new Entry { IsPassword = true }
            }
        };

        loginButton = new Button { Text = "Login",
BackgroundColor = Color.FromRgb(0, 148, 255) };
        layout.Children.Add(loginButton);

        Content = layout;
        loginButton.Clicked += (sender, e) =>
        {
            Debug.WriteLine("Clicked !");
        };
    }
}
```

تحتوي الصفحة على حاوية تخطيط layout container وملصقين اثنين two labels وحقلي إدخال two entries وزر button كما تعالج التعليمة البرمجية الحدث Clicked للزر، هناك أيضًا بعض خصائص التصميم المحددة حسب العناصر في الصفحة، في وقت التشغيل، على جهاز Android تبدو الصفحة كما يلي:



على الرغم من أن الصفحة تتمتع بتصميم بسيط، إلا أنها مزيج من السلوك والتصميم في نفس الملف.

يبدو تخطيط الصفحة نفسه المعرف باستخدام XAML كما يلي:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiXaml.Page1"
              BackgroundColor="#512bdf">

    <VerticalStackLayout Margin="15" Padding="30, 60, 30, 30">
        <Label Text="Please log in" FontSize="30"
TextColor="AntiqueWhite"/>
        <Label Text="Username" TextColor="White" />
        <Entry />
        <Label Text="Password" TextColor="White" />
        <Entry IsPassword="True" />
        <Button Text="Log in" BackgroundColor="#0094FF"
Clicked="LoginButton_Clicked" />
    </VerticalStackLayout>
</ContentPage>
```

تبدو التعليمات البرمجية C# التي تهيئ الصفحة وتنفذ معالج الأحداث لحدث Clicked لعنصر LoginButton في ملف التعليمات البرمجية في الخلفية كما يلي:

```
namespace MauiXaml;

public partial class Page1 : ContentPage, IPage
{
    public Page1()
    {
        InitializeComponent();
    }

 void LoginButton_Clicked(object sender, EventArgs e)
    {
        Debug.WriteLine("Clicked !");
    }
}
```

ملحوظة:

تقرأ الأسلوب InitializeComponent في منشئ الصفحة وصف XAML الخاص بالصفحة، ويحمل عناصر التحكم المختلفة الخاصة بالصفحة، ويحدد خصائصها، يمكن استدعاء هذا الأسلوب فقط إذا قمت بتعريف صفحة باستخدام ترميز XAML

لا يستدعي المثال السابق الذي يوضح كيفية إنشاء واجهة المستخدم باستخدام C# الأسلوب InitializeComponent

تسمح هذه البنية بفصل التصميم والسلوك، يتم تضمين إعلان واجهة المستخدم بالكامل في ملف مصدر مخصص واحد، وهو منفصل عن سلوك واجهة المستخدم، بالإضافة إلى ذلك، توفر علامات XAML وضوحًا أكبر للمطور الذي يحاول فهم مظهر التطبيق وأسلوبه.

ميزات استخدام XAML

يسمح لك استخدام XAML بفصل منطق السلوك عن تصميم واجهة المستخدم، يساعدك هذا الفصل على بناء كل قطعة بشكل مستقل، ويجعل التطبيق بأكمله أسهل في الإدارة أثناء نموه.

يتيح هذا النهج أيضًا لمصمم واجهة المستخدم المتخصص، العمل على تحديث مظهر واجهة المستخدم، وأسلوب عرضها، باستخدام أدوات تحرير XAML بشكل منفصل عن المطور الذي يقوم بتحديث منطق واجهة المستخدم.

اختبار بسيط:

١- ما هو الأسلوب التي يجب عليك استدعاؤها في منشئ الصفحة constructor لقراءة وصف XAML الخاص بالصفحة، وتحميل عناصر التحكم المختلفة على تلك الصفحة، وتعيين خصائصها؟

٢- أي من هذه المزايا يعد ميزة رئيسية لاستخدام XAML لتعريف واجهة المستخدم الخاصة بك؟

- تعمل XAML بشكل أكثر كفاءة من تعليمات C# لإنشاء واجهة المستخدم في وقت التشغيل.
- يمكن فصل تعريف واجهة المستخدم عن منطق التطبيق.
- XAML هي أكثر قابلية للنقل من التعليمات البرمجية C#

حل الاختبار:

١- InitializeComponent

٢- يمكن فصل تعريف واجهة المستخدم عن منطق التطبيق.

٣ الأنواع والخصائص Types and properties في .NET MAUI XAML.

XAML هي لغة ترميز تعريفية declarative markup مصممة بهدف تبسيط عملية إنشاء واجهة المستخدم الخاصة بك، تمثل العناصر the elements في XAML بشكل مباشر إنشاء الكائنات instantiation of objects التي يمكنك الوصول إليها في ملفات التعليمات البرمجية في الخلف.

في هذا الدرس، ستتعلم كيفية استخدام الأنواع types المتوفرة في XAML وكيفية تعيين خصائص هذه الأنواع وقراءتها set and read properties of these types

أين يتم تعريف الأنواع؟ types defined

يقوم .NET MAUI بتنفيذ محلل XAML الذي يقوم بتحليل العناصر المعلنة declared XAML elements وينشئ مثيل لكل عنصر كنوع .NET type. **إن لهجة XAML التي يفهمها محلل .NET MAUI خاصة بـ .NET MAUI.** على الرغم من أنها مشابهة لأطر عمل XAML الأخرى المستخدمة، مثل Windows Presentation Foundation

يتم تنفيذ .NET type التي تنفذ العناصر التي تحددها تعليمات XAML بواسطة التعليمات في العديد من تجميعات .NET. يتم تضمين العديد من هذه التجميعات كجزء من قوالب .NET MAUI. يمكنك أيضًا الاستفادة من أنواع مخصصة أخرى عن طريق تحميل التجميعات المناسبة كجزء من مشروعك، تتوفر العديد من التجميعات كحزم NuGet معظم الأنواع الشائعة التي يستخدمها تطبيق MAUI موجودة في **Microsoft.Maui.Dependencies** & **Microsoft.Maui.Extensions** packages

يتم تعريف كل نوع في مساحة اسم namespace في تعليمات XAML يمكنك تحديد مساحات الأسماء للأنواع namespaces for the types التي أنت تشير إليها، توجد معظم عناصر تحكم MAUI في مساحة اسم Microsoft.Maui.Controls بينما تحدد مساحة اسم Microsoft.Maui أنواع الأدوات المساعدة مثل Thickness وتتضمن مساحة اسم Microsoft.Maui.Graphics أنواعًا معممة مثل Color خيار تقديم الأنواع بهذه الطريقة يبرز قابلية توسيع XAML تتيح لك XAML إنشاء واجهة مستخدم لتطبيقك مع حرية تضمين .NET MAUI XAML elements, .NET types, and custom types. في الغالب، لا داعي للقلق بشأن مساحات الأسماء هذه حيث يتم إحضارها باستخدام ميزة الاستخدام الضمني C#'s implicit usings والتي تضيفها تلقائيًا على مستوى التطبيق.

كيفية إنشاء مثيل لأنواع instantiate types in XAML

الخطوة الأولى في استخدام XAML لإنشاء واجهة مستخدم هو إنشاء مثيل لأنواع عناصر التحكم في واجهة المستخدم، في XAML يمكنك إنشاء كائن من نوع محدد باستخدام بناء جملة عنصر العنصر، بناء جملة عنصر العنصر هو بناء جملة XML قياسي ومصمم بشكل جيد لإعلان عنصر، على سبيل المثال، إذا كنت تريد إنشاء ملصق أو تسمية Label بلون معين، فسيبدو عنصر XAML الخاص بك مثل التعليمات التالية:

```
<Label TextColor="AntiqueWhite"/>
```

محلل .NET MAUI XAML. يوزع عنصر XAML هذا لإنشاء مثيل للعنصر في الذاكرة، بشكل فعال، تسمية XAML التي تم تحليلها هي نفس التعليمة البرمجية C# التالية:

```
var myLabel = new Label
{
  TextColor = Color.FromRgb(255, 255, 100)
};
```

ما هي مساحة اسم XAML namespace

تذكر أنه لكي يتمكن محلل XAML بتحليل تعريف XAML لعنصر تحكم في صفحة بنجاح، يجب أن يكون لديه حق الوصول إلى التعليمات البرمجية التي تنفذ عنصر التحكم وتحدد خصائصه. يتم تنفيذ عناصر التحكم المتاحة لصفحة .NET MAUI في مجموعة من التجميعات المثبتة كجزء من حزمة Microsoft.Maui NuGet توجد عناصر التحكم في مساحة اسم .NET namespace. في هذه التجميعات.

في التعليمات البرمجية C# يمكنك إحضار مساحة اسم إلى النطاق باستخدام التوجيه using في صفحة XAML يمكنك الرجوع إلى مساحة اسم السمة xmlns attribute للصفحة، تظهر التعليمات البرمجية التالية مساحات الأسماء التي تستخدمها صفحة XAML التي تم إنشاؤها في الوحدة السابقة:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             ...>

    ...
</ContentPage>
```

`http://schemas.microsoft.com/dotnet/2021/maui` مساحة الاسم الأولي، هي مساحة الاسم الافتراضية للصفحة the page's default namespace

نموذج URI هذا لمساحة الاسم هو نموذج XML ويبدو مختلفاً إلى حد ما عن تلك التي قد تكون على دراية بها في C# ومع ذلك، فإن URI هذا هو ببساطة اسم مستعار لواحد أو أكثر من مساحات الأسماء المعرفة بواسطة التجميعات في حزمة NuGet **Microsoft.Maui** لذا فإن تحديد مساحة الاسم هذه في بداية الصفحة يجلب جميع أنواع وعناصر تحكم .NET MAUI types and controls إلى النطاق، إذا حذفت مساحة الاسم هذه، فلن تتمكن من استخدام عناصر التحكم مثل Button, Label, Entry, or StackLayout

مساحة الاسم الثانية http://schemas.microsoft.com/winfx/2009/xaml تشير إلى التجميعات التي تحتوي على الأنواع المضمنة المختلفة، مثل السلاسل والأرقام والخصائص .NET intrinsic types such as strings, numerics, and properties

في تعليمات XAML السابقة، تم تعيين الاسم المستعار **x** لمساحة الاسم هذه، يمكنك الرجوع إلى الأنواع الموجودة في مساحة الاسم هذه، في تعليمات XAML لهذه الصفحة، عن طريق إضافة بادئة **x:** إليها، على سبيل المثال، يتم تحويل كل صفحة XAML برمجياً إلى فئة class وتحدد اسم الفئة التي تم إنشاؤها باستخدام سمة الصفحة **x:Class** attribute of the page

```
<ContentPage ...
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="MauiXaml.Page1"
     ...>

    ...
</ContentPage>
```

يمكنك الرجوع إلى الأنواع في التجميعات الخاصة بك في تعليمات XAML من خلال مساحة اسم XAML namespace على سبيل المثال، إذا كان لديك أنواع وأساليب تريد استخدامها في تعليمات XAML المعرفة في مساحة اسم تسمى Utils في مشروعك، يمكنك إضافة مساحة اسم Utils إلى الصفحة كما هو موضح في التعليمات البرمجية التالية، في هذا المثال، يمكنك الوصول إلى الأنواع في مساحة الاسم هذه بإضافة بادئة لها بالاسم المستعار mycode

```
<ContentPage ...
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
     xmlns:mycode="clr-namespace:Utils"
             ...>
    ...
</ContentPage>
```

ملاحظة: سترى المزيد من الأمثلة على هذه التقنية لاحقاً في هذه الوحدة.

كيفية تحديد قيم الخصائص specify property values in XAML

في XML يمكنك استخدام السمات attributes لوصف عنصر أو توفير معلومات عنه، في XAML يمكنك استخدام السمات attributes لتعيين خصائص على النوع الأساسي underlying type على سبيل المثال، مع الأخذ في الاعتبار التعليمة البرمجية C# التالية:

```
var label = new Label { Text = "Username", TextColor = Color.Black };
```

تنشئ هذه العبارة كائناً جديدًا Label وتعين خصائص Text and TextColor لتعيين الخصائص في XAML يمكنك استخدام السمات attributes تبدو تعليمات XAML المقابلة كما يلي:

```
<Label Text="Username" TextColor="Black" />
```

أحد الأشياء التي قد تلاحظها والتي تختلف التعليمات XAML عن تعليمات C# هي قيم الخصائص property values على سبيل المثال، في التعليمات البرمجية C# يمكنك استخدام نوع Color type للخاصية TextColor ومع ذلك، في تعريف XAML يمكنك تعيين TextColor بقيمة سلسلة نصية string value وذلك لأن السلسلة هي العنصر الوحيد الصالح الذي يمكنك استخدامه ل XML attribute value لذلك، يجب أن تكون هناك طريقة لتحويل كل قيمة نصية إلى نوعها الصحيح. في XAML يمكنك إجراء هذا التحويل باستخدام محول النوع Type Converter

ما هو محول النوع؟ type converter

يقوم محول النوع بتحويل XML attribute المحددة كقيمة سلسلة string value إلى نوعها الصحيح، لفهم هذا المفهوم بشكل أفضل، ضع في اعتبارك المثال التالي:

```
<Label Text="Username" TextColor="Black" FontSize="42" FontAttributes="Bold,Italic" />
```

تقوم هذه التعليمة بإنشاء Label وتعيين خصائص Text, TextColor, FontSize, and FontAttributes properties

ابدأ بالخاصية الأولى Text عبارة عن سلسلة نصية string مما يعني أن صفحة XAML لا تحتاج إلى محول نوع type converter بعد ذلك TextColor يستخدم النوع Color لذلك يتطلب XAML محول نوع لترجمة السلسلة string إلى القيمة

Color المقابلة، خاصية FontSize عبارة عن عدد صحيح integer لذا يتطلب XAML محول نوع لتحليل السلسلة إلى عدد صحيح.

أخيرًا FontAttributes هو مثال على كائن معقد complex object يمكنك دمج القيم كسلسلة محددة بفاصلة "Bold,Italic" يتم التعامل مع السلسلة المحددة بفاصلة comma-delimited string على أنها تعدادًا قائماً على [Flags] وسيقوم محول النوع المناسب بتطبيق OR لكل بت من القيمة على الخاصية the bitwise OR of the value to the property

يحتوي .NET MAUI على محولات نوع لمعظم الفئات المضمنة built-in classes ويستخدم محولات النوع هذه تلقائياً، ومع ذلك، إذا لم يكن هناك محول معين، يمكنك كتابة المحول الخاص بك وإقرانه بالنوع الخاص بك لجعله قابلاً للاستخدام في XAML

تعيين نوع معقد Complex type assignment

تعد محولات النوع Type converters رائعة لإعدادات الخصائص البسيطة؛ ومع ذلك، في بعض الحالات، تحتاج إلى إنشاء كائن كامل بقيم خصائصه الخاصة به، حل هذه المشكلة هو تغيير تعيين الخاصية لاستخدام بناء جملة يستند إلى العناصر element-based syntax يُسمى هذا البناء النحوي نموذج عنصر الخاصية **Property Element** يتضمن بناء الجملة هذا تقسيم مُعيّن الخاصية إلى نموذج الأصل والتابع أو الأب والطفل parent-child حيث يتم التعبير عن الخاصية في علامة عنصر من النموذج Type.PropertyName افترض أنك تريد تعيين أداة مُعرّف إيماءة gesture recognizer إلى label بحيث يمكن لمستخدم التطبيق الضغط على label يُعد مُعرّف الإيماءة كائنًا معقدًا له خصائصه الخاصة. عادةً، يلزم تعيين هذه الخصائص لضمان الأداء الوظيفي السليم:

```
<TapGestureRecognizer NumberOfTapsRequired="2" />
```

إذا كنت بحاجة إلى تعيين هذه القيمة إلى Label يمكنك كتابة XAML كما يلي:

```
<Label Text="Username" TextColor="Black"
FontSize="42" FontAttributes="Bold,Italic">

  <Label.GestureRecognizers>
    <TapGestureRecognizer NumberOfTapsRequired="2" />
  </Label.GestureRecognizers>

</Label>
```

يحتوي Label type على خاصية تسمى GestureRecognizers باستخدام نموذج Property Element يمكنك إضافة TapGestureRecognizer إلى قائمة الإيماءات ل Label

خاصية المحتوى الافتراضية Default content property

تحتوي بعض عناصر تحكم .NET MAUI على خاصية محتوى افتراضية default content property تسمح لك هذه الخاصية بتحديد قيمة خاصية على أساس عنصر تحكم دون الإشارة إليه صراحةً في XAML ألق نظرة على جزء XAML التالي:

```
<VerticalStackLayout>
    <VerticalStackLayout.Children>
        <Label Text="Please log in" />
    </VerticalStackLayout.Children>
</VerticalStackLayout>
```

تقوم التعليمات بإنشاء VerticalStackLayout وإضافة Label كعنصر تابع، نظرًا لأنه من الشائع إضافة عناصر تابعة إلى VerticalStackLayout فإن خاصية `Children` الخاصة به هي خاصية المحتوى الافتراضية، وهذا يعني أنه يمكنك إضافة عنصر تابع، دون تحديد العناصر الفرعية `Children` صراحةً على النحو التالي:

```
<VerticalStackLayout>
    <Label Text="Please log in" />
</VerticalStackLayout>
```

اختبار بسيط:

١- ماذا يفعل محول النوع Type Converter

٢- كيف يمكنك إحضار أنواعك المخصصة custom types إلى نطاق ملف XAML

حل الاختبار:

١- يتم استخدام محول النوع Type Converter لتحويل قيمة سمة XML attribute value المحددة كسلسلة string إلى نوعها الصحيح.

٢- استخدم سمة xmlns للصفحة xmlns attribute of the page وامنح مساحة الاسم namespace التي تحتوي على الأنواع المخصصة اسمًا مستعارًا يمكنك الرجوع إليه من عناصر تحكم XAML controls على الصفحة.

٤ معالجة الحدث Event handling في XAML

بعد إنشاء واجهة مستخدم XAML يمكنك إضافة تعليمات برمجية للاستجابة للتفاعلات التي تحدث أثناء زيارة المستخدم للصفحة، يقوم .NET MAUI بإعلام التطبيق بإدخال المستخدم وتفاعلاته من خلال الأحداث القياسية .NET standard events

في هذا الدرس، ستتعلم كيفية التعامل مع هذه الأحداث وتنفيذ العمليات المتوقعة من قبل المستخدم.

تسمية العناصر في صفحة Naming elements in a XAML page

غالبًا ما تحتاج التعليمات البرمجية لمعالجة الأحداث المتكررة، إلى الإشارة إلى عناصر تحكم معينة وخصائصها على صفحة، لذا يمكنك تعيين اسم فريد لكل عنصر تحكم، للقيام بذلك، استخدم XAML attribute x:Name تنفذ السمة x:Name أمرين:

- يضيف حقلاً خاصًا إلى ملف التعليمات الذي تم إنشاؤه والذي تم تعيينه لهذا العنصر، استخدم هذا الحقل في التعليمات للتفاعل مع العنصر المرئي لتعيين خصائص وقت التشغيل ومعالجة الأحداث.
- يجعل العنصر معروفًا لـ XAML من خلال هذا الاسم، يمكنك الرجوع إلى هذه العناصر من عناصر أخرى محددة في نفس ملف XAML

لا يمكنك استخدام أي نص عشوائي عند تسمية العنصر، لأنه يتم استخدام القيمة المعينة للسمة x:Name لإنشاء حقل في التعليمات البرمجية، بدلاً من ذلك، يجب أن يتوافق مع اصطلاحات التسمية المتعارف عليها لمتغير، يجب أن يكون الاسم فريدًا أيضاً، لأنه مُجمَّع في تعريف التعليمات البرمجية في الخلف.

بعد توفير اسم لعنصر معين، يمكنك التفاعل مع هذا العنصر في ملف التعليمات البرمجية، يعرف جزء XAML التالي عنصر Label يسمى CounterLabel تم أخذ هذا المثال من التطبيق الافتراضي الذي ينشئه قالب .NET MAUI

```
    <Label Text="Current count: 0"
           ...
           x:Name="CounterLabel"
           ... />
```

في التعليمات البرمجية الخلفية لهذه الصفحة، يمكنك الرجوع إلى عنصر التحكم هذا من خلال حقل CounterLabel وتعديل خصائصه:

```
count++;
```

```
CounterLabel.Text = $"Current count: {count}";
```

هام:

لن تتم تهيئة الحقل حتى يتم تشغيل أسلوب InitializeComponent للصفحة، هذا الأسلوب هو جزء من عملية تحليل XAML وإنشاء مثيل للكائنات، ضع أي تعليمة برمجية تتفاعل مع عنصر معرّف في XAML بعد هذا الاستدعاء.

الاستثناء لهذه القاعدة هو ContentPage class نفسه، يمكنك الوصول إلى كافة الخصائص في الفئة، قبل تنفيذ الأسلوب InitializeComponent ومع ذلك، إذا قمت بتعيين أي خصائص على هذه الفئة في XAML فإن قيم هذه الخصائص ستستبدل أي قيم قد ربما قمت بتعيينها قبل تنفيذ InitializeComponent

استخدام سمة لربط الأحداث attribute to wire up events

تعرض العديد من عناصر التحكم الخصائص التي تتوافق مع الأحداث التي يمكن أن تستجيب لها عناصر التحكم هذه، مثل الحدث Clicked لزر، تدعم عناصر التحكم المختلفة مجموعات متنوعة من الأحداث، على سبيل المثال، يمكن لعنصر Button الاستجابة لأحداث Clicked, Pressed, and Released events بينما يحتوي عنصر Entry على أحداث مثل TextChanged يمكنك تهيئة خاصية حدث في ترميز XAML الخاص بصفحة، وتحديد اسم الأسلوب الذي سيتم تشغيله عند تشغيل الحدث.

يجب أن يفي أسلوب الحدث متطلبات التوقيع التالية:

- لا يمكن إرجاع قيمة؛ يجب أن يكون الأسلوب void
- يجب أن يأخذ معامِلين two parameters مرجع كائن reference `object` يشير إلى الكائن الذي تسبب في الحدث (المعروف باسم المرسل known as) (the `sender`) ومعلمة `EventArgs parameter` التي تحتوي على أي وسيطات تم تمريرها إلى معالج الحدث بواسطة المرسل.
- يجب أن يكون معالج الحدث خاصًا `private` لا يتم فرض هذا، ولكن إذا جعلت معالج الحدث عاماً، فإنه يصبح متاحًا للعالم الخارجي، ويمكن أن يستدعيه إجراء آخر غير الحدث المتوقع أن يستدعيه.
- يمكن أن يكون معالج الحدث غير متزامن `async` إذا كان يحتاج إلى تشغيل عمليات غير متزامنة asynchronous

يوضح المثال التالي تعريف معالج الحدث Clicked للزر في تطبيق الافتراضي من قالب .NET MAUI. يتبع اسم الأسلوب اصطلاحا قياسياً؛ On متبوعة باسم عنصر

التحكم (يُسمى Counter) واسم الحدث (Clicked) لا يتم فرض هذه الاتفاقية، ولكنها ممارسة جيدة:

```
private void OnCounterClicked(object sender,
EventArgs e)
{
    ...
}
```

فصل الاهتمامات Separation of Concerns

ربط الأحداث في XAML مريح، ولكنه يخلط بين سلوك عنصر التحكم وتعريف واجهة المستخدم، يفضل العديد من المطورين إبقاء هذه العناصر مميزة وإجراء جميع اشتراكات معالجات الأحداث في التعليمات البرمجية الخلفية للعناصر المسماة. من الأسهل معرفة ما تم ربطه ومكان تعيين السلوك. أيضاً هذا النهج يجعل من الصعب قطع التعليمات بالخطأ عن طريق إزالة معالج في XAML دون إدراك ذلك، لن يلتقط المحول البرمجي معالجاً تمت إزالته، ولن يصبح مشكلة إلا عندما لا تنفذ التعليمات البرمجية هذا السلوك بشكل صحيح.

يعد اختيار ربط معالجات الأحداث باستخدام XAML أو باستخدام التعليمات البرمجية مسألة اختيار شخصي.

لربط معالج حدث في التعليمات البرمجية استخدم عامل التشغيل `+=` للاشتراك في الحدث، عادة ما تقوم بتنفيذ هذه العملية في الدالة الإنشائية للصفحة، بعد استدعاء InitializeComponent

```
public partial class MainPage : ContentPage, IPage
{
    public MainPage()
    {
        InitializeComponent();
        Counter.Clicked += OnCounterClicked;
    }
    ...
   private void OnCounterClicked(object sender,
EventArgs e)
    {
        ...
    }
}
```

ملاحظة:

يمكنك استخدام هذا النهج للاشتراك في أساليب معالجة أحداث متعددة event-handling methods لنفس الحدث، يتم تشغيل كل أسلوب معالجة حدث عند وقوع الحدث، على الرغم من أنه لا ينبغي لك أن تفترض أنها ستنفذ بأي ترتيب معين، لذا لا تقم بإدخال أي تبعيات بينها.

وبالمثل، يمكنك إزالة معالج أحداث عن طريق إلغاء اشتراكه مع الحدث باستخدام عامل التشغيل `-=` لاحقاً في تطبيقك:

```
Counter.Clicked -= OnCounterClicked;
```

اختبار بسيط:

١- ما هي المعلمات parameters التي تم تمريرها إلى أسلوب معالجة الأحداث event-handling method

- اسم الحدث واسم عنصر التحكم الذي أستدعى الحدث.
- مرجع إلى صفحة XAML وعنصر التحكم على الصفحة التي استدعت الحدث.
- مرجع إلى عنصر التحكم الذي أستدعى الحدث، وكائن EventArgs الذي يحتوي على أي معلومات إضافية مطلوبة لمعالجة.

٢- ما عامل التشغيل الذي يمكنك استخدامه لربط معالج الأحداث بحدث في C#

حل الاختبار:

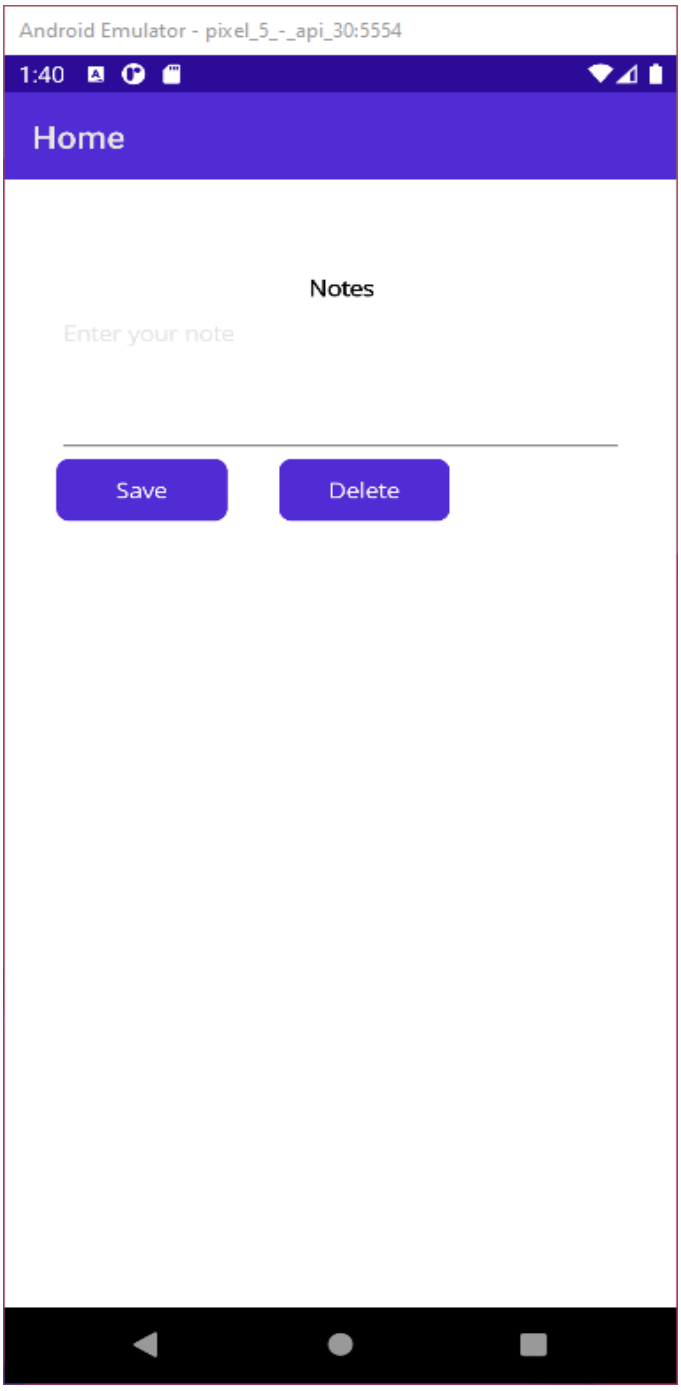
١- مرجع إلى عنصر التحكم الذي أستدعى الحدث، وكائن EventArgs الذي يحتوي على أي معلومات إضافية مطلوبة لمعالجة

٢- استخدم العامل **+=**

٥ تمرين: إنشاء أول صفحة XAML

يقوم المهندسون من شركة الأدوات المساعدة للطاقة بانتظام بزيارات العملاء لإصلاح الأجهزة، وتنفيذ مهام الصيانة الكهربائية الأخرى، يسمح جزء من التطبيق للمهندس بتدوين ملاحظات حول الزيارة، يعرض محرراً بسيطاً حيث يمكن للمهندس إدخال التفاصيل وحفظها.

يبدو التطبيق على Android كما يلي:



لقد طلب منك إضافة بعض الميزات الإضافية إلى هذه الصفحة، قبل البدء، تريد فهم كيفية إنشاء الصفحة، حتى تنظر إلى التعليمات البرمجية المصدر، لاحظت أنه تم إنشاء واجهة المستخدم بالكامل باستخدام التعليمات البرمجية C# على الرغم من أن هذا النهج يعمل، فإنه يخلط بين التعليمات البرمجية التي تتعامل مع التخطيط ومع التعليمات البرمجية التي تتحكم في الطريقة التي تعمل بها واجهة المستخدم، تدرك أنه قبل وقت طويل، هناك خطر من أن يصبح جانبا التطبيق مؤمنين معاً، مما يجعل الصيانة المستقبلية صعبة، وربما تجعل التطبيق أكثر هشاشة مع إضافة المزيد من الميزات. قررت فصل تصميم واجهة المستخدم عن منطق واجهة المستخدم عن طريق استخراج التعليمات البرمجية C# التي تحدد التخطيط من التطبيق واستبداله بصفحة XAML

تستخدم هذه الوحدة .NET 9.0 SDK. تأكد من تثبيت .NET 9.0 عن طريق تشغيل الأمر التالي في الوحدة الطرفية للأوامر command terminal

```
dotnet --list-sdks
```

يظهر إخراج مشابه للمثال التالي:

```
8.0.100 [C:\Program Files\dotnet\sdk]
9.0.100 [C:\Program Files\dotnet\sdk]
```

تأكد من إدراج إصدار يبدأ بـ 9 إذا لم يتم سرد أي منها أو لم يتم العثور على الأمر، فقم بتثبيت أحدث اصدار .NET 9.0 SDK

مراجعة التطبيق الموجود

١- نسخ مستودع GitHub لهذا التمرين محلياً على جهاز الكمبيوتر.

ملحوظة: من الأفضل نسخ محتوى التمرين أو تنزيله إلى مسار مجلد قصير، مثل C:\dev لتجنب تجاوز الملفات التي تم إنشاؤها الحد الأقصى لطول المسار، وينتج خطأ.

٢- انتقل إلى مجلد exercise1 في نسختك من المستودع.

٣- افتح ملف Notes.sln Visual Studio في هذا المجلد أو المجلد في Visual Studio Code

٤- في نافذة Solution Explorer قم بتوسيع مشروع Notes وقم بتوسيع الملف MainPage.xaml وافتح ملف MainPage.xaml.cs

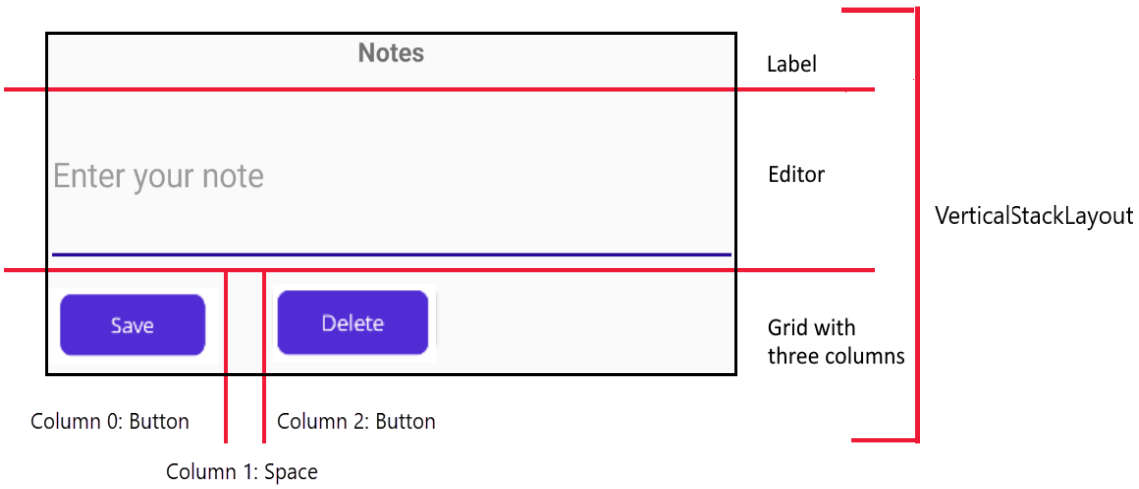
٥- راجع فئة MainPage class المحددة في هذا الملف. يحتوي المنشئ على التعليمات البرمجية التالية التي تنشئ واجهة المستخدم:

```
public MainPage()
```

```
{
    var notesHeading = new Label() { Text = "Notes",
HorizontalOptions = LayoutOptions.Center,
FontAttributes = FontAttributes.Bold };

    editor = new Editor() { Placeholder = "Enter your
note", HeightRequest = 100 };
    editor.Text = File.Exists(_fileName) ?
File.ReadAllText(_fileName) : string.Empty;

    var buttonsGrid = new Grid() { HeightRequest =
40.0 };
    buttonsGrid.ColumnDefinitions.Add(new
ColumnDefinition() { Width = new GridLength(1.0,
GridUnitType.Auto) });
    buttonsGrid.ColumnDefinitions.Add(new
ColumnDefinition() { Width = new GridLength(30.0,
GridUnitType.Absolute) });
    buttonsGrid.ColumnDefinitions.Add(new
ColumnDefinition() { Width = new GridLength(1.0,
GridUnitType.Auto) });

    var saveButton = new Button() { WidthRequest =
100, Text = "Save" };
    saveButton.Clicked += OnSaveButtonClicked;
    Grid.SetColumn(saveButton, 0);
    buttonsGrid.Children.Add(saveButton);

    var deleteButton = new Button() { WidthRequest =
100, Text = "Delete" };
    deleteButton.Clicked += OnDeleteButtonClicked;
    Grid.SetColumn(deleteButton, 2);
    buttonsGrid.Children.Add(deleteButton);

    var stackLayout = new VerticalStackLayout
    {
        Padding = new Thickness(30, 60, 30, 30),
        Children = { notesHeading, editor,
buttonsGrid }
    };
    this.Content = stackLayout;
}
```

تتكون واجهة المستخدم من VerticalStackLayout يحتوي على Label, Editor وشبكة Grid بها ثلاثة أعمدة columns يحتوي العمود الأول على عنصر التحكم saveButton ويحتوي العمود الثاني على فاصل spacer ويحتوي العمود الثالث على عنصر التحكم deleteButton

يوضح الرسم التخطيطي التالي بنية واجهة المستخدم:



لاحظ أن فئة MainPage تحتوي أيضا على أساليب معالجة الأحداث للأزرار وبعض التعليمات البرمجية التي تهيئ عنصر Editor هذه التعليمة البرمجية غير مميزة عن تعريف واجهة المستخدم.

٦- قم بإنشاء التطبيق وتشغيله Build and run على Windows فقط لمعرفة شكله، حدد F5 لإنشاء التطبيق وتشغيله.

٧- أغلق التطبيق وارجع إلى Visual Studio أو Visual Studio Code عند الانتهاء.

إنشاء إصدار XAML من واجهة المستخدم XAML version of the UI

١- افتح الملف MainPage.xaml تمثل العلامات markup في هذه الصفحة صفحة محتوى MAUI فارغة:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       x:Class="Notes.MainPage">

</ContentPage>
```

٢- إضافة عنصر VerticalStackLayout تحكم إلى content page

```
<ContentPage ...>
    <VerticalStackLayout Margin="30,60,30,30">

    </VerticalStackLayout>
</ContentPage>
```

٣- أضف Label إلى VerticalStackLayout وعيين خصائصه Text, HorizontalTextAlignment, FontAttributes كما هو موضح أدناه:

```
<ContentPage ...>
    <VerticalStackLayout ...>
        <Label Text="Notes"
               HorizontalOptions="Center"
               FontAttributes="Bold" />
    </VerticalStackLayout>
</ContentPage>
```

٤- أضف عنصر Editor إلى VerticalStackLayout

```
<ContentPage ...>
    <VerticalStackLayout ...>
        <Label .../>

        <Editor x:Name="editor"
                Placeholder="Enter your note"
                HeightRequest="100" />
    </VerticalStackLayout>
</ContentPage>
```

٥- إضافة Grid تابع إلى VerticalStackLayout يجب أن يحتوي هذا Grid على ثلاثة أعمدة columns يتم تغيير حجم العمودين الأول والثالث تلقائياً، بينما يبلغ عرض العمود الثاني 30

```
<ContentPage ...>
    <VerticalStackLayout ...>
        <Label .../>
        <Editor .../>

        <Grid ColumnDefinitions="Auto, 30, Auto">
        </Grid>

    </VerticalStackLayout>
</ContentPage>
```

٦- أضف Button إلى العمود الأول من Grid سيكون زر الحفظ Save button

```
<ContentPage ...>
    <VerticalStackLayout ...>
        <Label .../>

        <Editor .../>

        <Grid ...>
            <Button Grid.Column="0"
                Text="Save"
                WidthRequest="100"
                Clicked="OnSaveButtonClicked" />
        </Grid>
    </VerticalStackLayout>
</ContentPage>
```

٧- أضف Button آخر إلى العمود الثالث من Grid سيكون زر الحذف Delete

```
    <Button Grid.Column="2"
        Text="Delete"
        WidthRequest="100"
        Clicked="OnDeleteButtonClicked" />
```

إزالة تعليمات التخطيط the layout code من ملف التعليمات الخلفي code-behind file

١- في نافذة Solution Explorer قم بتوسيع عقدة MainPage.xaml وافتح ملف MainPage.xaml.cs

٢- قم بإزالة Editor field من فئة MainPage class

٣- في ملف MainPage.xaml.cs في الدالة الإنشائية MainPage constructor قم بإزالة جميع التعليمات التي تنشئ عناصر واجهة المستخدم، واستبدلها باستدعاء الأسلوب InitializeComponent أضف التعليمات التي تتحقق من وجود الملف المستخدم لتخزين الملاحظات store the notes exists وإذا كان الأمر كذلك، فستقرأ المحتويات، وتملأ الحقل النصي Text field لعنصر Editor يجب أن تبدو الدالة الإنشائية كما يلي:

```
public partial class MainPage : ContentPage
{
```

```
  string _fileName =
Path.Combine(FileSystem.AppDataDirectory,
"notes.txt");

 public MainPage()
 {
      InitializeComponent();

    if (File.Exists(_fileName))
    {
       editor.Text = File.ReadAllText(_fileName);
    }
  }

    ...
}
```

٤- في قائمة Build حدد Rebuild Solution تحقق من أن التطبيق يبني دون أي أخطاء.

٥- قم بتشغيل التطبيق، يجب أن يعمل كما كان من قبل تماماً.

٦- إذا كان لديك الوقت، فقم بنشر التطبيق وتشغيله باستخدام محاكي Android يجب أن تشبه واجهة مستخدم التطبيق ذلك الموضح في الصورة في بداية هذا التمرين.

٦ ملحقات علامات أو ترميز XAML markup extensions

سيتم تحديد الكثير من تعريف XAML الخاص بك في وقت التحويل البرمجي، غالباً ما تعرف مكان وضع العناصر elements والألوان والخطوط colors and fonts التي سيتم استخدامها، والقيم الحرفية literal values التي يجب تعيينها للخصائص.

ومع ذلك، أحياناً تحتاج إلى تعيين قيمة خاصية إلى قيمة لا يمكن تحديدها في وقت التحويل البرمجي، ولا تُعرَف هذه القيم إلا عند تشغيل البرنامج. وفي هذه المواقف، يمكنك إنشاء كائن يوفر قيمة لـ XAML في وقت التشغيل، يدعم XAML ملحقات العلامات Markup Extensions لهذا الغرض.

في هذا الدرس، ستتعلم كيفية إنشاء ملحقات العلامات واستخدامها.

ما هو ملحق العلامات ?What is a markup extension

ملحق العلامات هو فئة تستخدمها في XAML للوصول إلى قيم وقت التشغيل. لنفترض أن لديك العديد من labels المعرفة في واجهة مستخدم XAML وتريد تعيين الخاصية FontSize إلى نفس القيمة في جميع أنحاء التطبيق، للتأكد من أن كل تصميم labels متناسق، يمكنك تعيين الخاصية FontSize باستخدام XAML كما هو موضح في المثال التالي:

```
<Label Text="Hello, World!"
       Grid.Row="0"
       SemanticProperties.HeadingLevel="Level1"
       FontSize="28"
       HorizontalOptions="CenterAndExpand"/>
```

يمكنك تكرار نفس الإعداد لكل Label ولكن ماذا لو أردت تغيير هذه القيمة لاحقًا؟ تحتاج إلى العثور على كل مثيل لهذه الخاصية وإجراء التغيير. افترض أيضًا أنك لا تعرف القيمة التي يجب استخدامها؛ يمكن حسابها في وقت التشغيل استنادًا إلى عوامل مثل اتجاه الجهاز أو دقة الشاشة أو اعتبارات أخرى، في هذه الحالات، تحتاج إلى شيء أكثر تعقيدًا من القيمة الحرفية ذات التعليمات البرمجية المضمنة hard-coded literal هنا يكون ملحق الترميز مفيدًا markup extension تمنحك ملحقات الترميز المرونة في كيفية الحصول على قيمة يتم استخدامها في XAML

إنشاء ملحق ترميز Creating a markup extension

ملحق العلامات markup extension هو فئة class تنفذ واجهة Microsoft.Maui.Controls.Xaml.IMarkupExtension interface تعرف هذه الواجهة أسلوباً واحدًا، يسمى ProvideValue بالتوقيع التالي:

```
public object ProvideValue(IServiceProvider
serviceProvider)
{
    ...
}
```

الغرض من هذا الأسلوب هو توفير قيمة لعلامات XAML markup لاحظ أن نوع الإرجاع هو object لذلك يمكن أن تكون القيمة من أي نوع طالما أنها مناسبة للمكان الذي يتم استخدامها فيه، على سبيل المثال، في ملحق العلامات الذي يقوم بحساب حجم خط font size وإرجاعه، يجب أن يكون نوع الإرجاع double

تحتوي المعلمة serviceProvider parameter على معلومات سياقية حول مكان استخدام ملحق العلامات في التعليمات البرمجية XAML ومن بين معلومات أخرى، تحدد عنصر التحكم الذي يتم تطبيق الملحق عليه.

يمكنك إبقاء ملحق العلامات للخاصية FontSize بسيطاً، في المثال التالي، تعرض فئة MainPage class حقل double field باسم MyFontSize تنفذ الفئة GlobalFontSizeExtension class الواجهة IMarkupExtension interface ويرجع الأسلوب ProvideValue قيمة المتغير MyFontSize

```
namespace MyMauiApp;

public partial class MainPage : ContentPage
{
    public const double MyFontSize = 28;

    public MainPage()
    {
        InitializeComponent();
        ...
    }
    ...
}

public class GlobalFontSizeExtension :
IMarkupExtension
{
   public object ProvideValue(IServiceProvider
serviceProvider)
    {
        return MainPage.MyFontSize;
```

```
    }
}
```

ملاحظة:

يجب أن يكون الحقل MyFontSize عضوا static في MainPage class للسماح بالإشارة إليه في الأسلوب ProvideValue method بهذه الطريقة. الممارسة الجيدة تعني أنه في هذه الحالة، يجب أن يكون المتغير ثابتاً أيضاً، القيمة const value هي static

يمكن للأسلوب ProvideValue method أيضاً إجراء تعديلات على القيمة التي تم إرجاعها، اعتمادًا على الاتجاه وعامل نموذج الجهاز.

تطبيق ملحق العلامات markup extension على عنصر تحكم control في XAML

لاستخدام ملحق العلامات في تعليمات XAML أضف مساحة الاسم التي تحتوي على `GlobalFontSizeExtension class` إلى قائمة مساحات الأسماء في `namespaces in the ContentPage` في المثال التالي، يتم إعطاء مساحة الاسم هذه الاسم المستعار mycode

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:mycode="clr-namespace:MyMauiApp"
        x:Class="MyMauiApp.MainPage">
```

يمكنك استخدام ملحق العلامات markup extension لتعيين الخاصية FontSize على هذا النحو، لاحظ أن الاصطلاح هو أن ملحق العلامات يحتوي على ملحق اللاحقة suffix Extension باسمه، يتعرف XAML على هذه اللاحقة suffix ولا تحتاج إلى تضمينها عند استدعاء الملحق من تعليمة XAML في المثال التالي، تتم الإشارة إلى `GlobalFontSizeExtension class` ببساطة على أنه `GlobalFontSize`

```
<Label Text="Hello, World!"
        Grid.Row="0"
        SemanticProperties.HeadingLevel="Level1"
        FontSize="{mycode:GlobalFontSize}"
        HorizontalOptions="CenterAndExpand"/>
```

يمكنك تطبيق ملحق العلامات نفسه في تعليمات XAML لأي عنصر تحكم يحتاج إلى تحديد حجم الخط font size في وقت لاحق، إذا قررت تغيير حجم الخط، فستحتاج فقط إلى تعديل تعريف المتغير MyFontSize في MainPage

The StaticExtension class

بالرغم من فائدة ملحق علامات GlobalFontSize فمن غير المرجح أن تقوم بإنشاء مثل هذا الملحق، والسبب في ذلك بسيط؛ حيث يوفر .NET MAUI بالفعل ملحقًا أكثر عمومية، يسمح لك بالإشارة إلى أي قيمة ثابتة في تعليماتك، يُسمى هذا الملحق StaticExtension أو Static للاختصار. تظهر التعليمات التالية المخطط التفصيلي الأساسي لفئة الملحق هذه:

```
[ContentProperty ("Member")]
public class StaticExtension : IMarkupExtension
{
    public string Member {get; set;}
    public object ProvideValue (IServiceProvider
serviceProvider)
    {
        ...
    }
}
```

ملاحظة:

الغرض من ملحقات العلامات المخصصة custom markup extensions هو السماح لك بمعالجة المواقف الأكثر تعقيدًا بدلاً من الحالة الثابتة البسيطة، على سبيل المثال، قد تحتاج إلى تغيير حجم الخط ديناميكياً استنادًا إلى عامل نموذج الجهاز.

لاستخدام هذه الفئة في XAML قم بتوفير اسم المتغير الثابت static variable الذي تريد الرجوع إليه في الخاصية Member ويرجع الأسلوب ProvideValue القيمة في هذا المتغير. يوضح المثال التالي كيفية استخدامه:

```
<Label Text="Hello, World!"
       Grid.Row="0"
       SemanticProperties.HeadingLevel="Level1"
   FontSize="{x:Static Member=mycode:MainPage.MyFontSize}"
       HorizontalOptions="CenterAndExpand"/>
```

يوفر .NET MAUI مجموعة من فئات ملحقات علامات أخرى markup extension classes والتي يمكنك استخدامها في سيناريوهات مثل ربط البيانات،

والإشارة إلى الموارد والأنماط الديناميكية، ومعالجة مجموعات البيانات data binding, referencing dynamic resources and styles, and handling arrays of data

اختبار بسيط:

١- ما هو ملحق العلامات أو الترميز markup extension الذي يسمح لك بتعيين خاصية XAML إلى قيمة ثابتة static value تم تعريفها في فئة التعليمات الخلفية code-behind class

٢- ما هي الواجهة interface التي تستخدمها لإنشاء ملحق علامات مخصص custom markup extension

حل الاختبار:

١- قم بالإشارة إليه كـ x:Static في XAML Static

٢- IMarkupExtension

٧ قيم خاصة بنظام التشغيل Platform-specific values في XAML

ستكون التجربة المرئية لتطبيقك مختلفة على كل نظام تشغيل، ستحتاج في كثير من الأحيان إلى ضبط واجهة المستخدم لكل نظام استنادًا إلى العناصر المرئية التي تستخدمها، يتيح لك .NET MAUI. إدارة تخطيط تطبيقك استنادًا إلى خصائص الجهاز هذه.

في هذا الدرس، ستتعرف على الميزات التي يوفرها .NET MAUI. للسماح لك بتعديل واجهة المستخدم لتطبيقاتك وفقاً لنظام التشغيل الذي تعمل عليه.

استخدام فئة الجهاز Using the Device class

فئة DeviceInfo class هي فئة أداة مساعدة توفر معلومات خاصة بالجهاز الذي يعمل عليه تطبيقك، وهي تعرض هذه المعلومات من خلال مجموعة من الخصائص، الخاصية الأكثر أهمية هي DeviceInfo.Platform والتي تعيد string تشير إلى نوع الجهاز قيد الاستخدام حاليًا: Android أو iOS أو WinUI أو macOS

ضع في اعتبارك السيناريو التالي كمثال على الظرف الذي ربما تستخدم فيه هذه الميزة، السلوك الافتراضي لتطبيق .NET MAUI iOS. هو أن المحتوى المضاف إلى الصفحة يتعدى على شريط حالة iOS في أعلى الشاشة، تريد تغيير هذا السلوك. أبسط حل هو نقل المحتوى لأسفل داخل الصفحة، يعالج مشروع Notes solution الذي أنشأته في التمرين السابق هذه المشكلة عن طريق تعيين خاصية Padding لعنصر VerticalStackLayout لنقل المحتوى لأسفل بمقدار 60 نقطة:

```
<VerticalStackLayout x:Name="MyStackLayout"
Padding="30,60,30,30">
    ...
</VerticalStackLayout>
```

المشكلة هي أن هذا النهج قابل للتطبيق فقط على iOS يؤدي نقل المحتوى إلى أسفل هذا القدر على Android وWinUI إلى إهدار مساحة الشاشة في أعلى الصفحة.

يمكنك الاستعلام عن الخاصية DeviceInfo.Platform لحل مشكلة العرض هذه. أضف التعليمات التالية إلى منشئ صفحة التطبيق app's page constructor لتوسيع المساحة في أعلى الصفحة، ولكن فقط لنظام التشغيل iOS

```
MyStackLayout.Padding =
 DeviceInfo.Platform == DevicePlatform.iOS
 ? new Thickness(30, 60, 30, 30) // Shift down by
60 points on iOS only
```

```
 : new Thickness(30); // Set the default margin to
be 30 points
```

ملاحظة:

`DevicePlatform.iOS` عبارة عن بنية `DevicePlatform struct` تعيد قيمة السلسلة لنظام التشغيل `string value iOS` توجد خصائص مكافئة للأنظمة المدعومة الأخرى. يجب عليك استخدام هذه الخصائص بدلاً من المقارنة بالسلاسل المبرمجة مسبقًا hard-coded strings إنها ممارسة جيدة، وهي تحمي تعليماتك في المستقبل إذا تغيرت بعض قيم السلسلة هذه في المستقبل.

تعمل هذه التعليمة البرمجية، ولكنها موجودة في ملف التعليمات الخلفي للصفحة، ترك المساحة أو التبطين padding هو قيمة خاصة بواجهة المستخدم، يمكن القول إنه من الأنسب والمريح القيام بذلك من XAML بدلاً من التعليمات البرمجية الخلفية.

استخدام ملحق العلامات OnPlatform markup extension

يوفر .NET MAUI XAML ملحق العلامات `OnPlatform` والذي يسمح لك باكتشاف نظام التشغيل من داخل XAML يمكنك تطبيق ملحق العلامات هذا كجزء من تعليمات XAML التي تعين قيمة الخاصية، يتطلب منك الملحق توفير نوع الخاصية، جنباً إلى جنب مع سلسلة من كتل `On Platform blocks` التي تقوم فيها بتعيين قيمة الخاصية وفقاً لنظام التشغيل.

ملاحظة:

إن ملحق العلامات `OnPlatform` عام؛ فهو يأخذ معلمة نوع type parameter ويضمن النوع الذي تحدده سمة `TypeArguments attribute` استخدام نوع الملحق extension type الصحيح.

يمكنك تعيين الخاصية Padding مثل هذا، لاحظ أن نوع الخاصية Padding هو Thickness

```
<VerticalStackLayout>
    <VerticalStackLayout.Padding>
       <OnPlatform x:TypeArguments="Thickness">
         <On Platform="iOS" Value="30,60,30,30" />
       </OnPlatform>
    </VerticalStackLayout.Padding>
    <!--XAML for other controls goes here -->
    ...
</VerticalStackLayout>
```

بالنسبة للأنظمة غير iOS سيبقى إعداد الحشو معيناً إلى قيمته الافتراضية "0,0,0,0" بالنسبة إلى WinUI وAndroid يمكنك تعيين الحشو إلى 30 نقطة padding to 30 points مع كتل إضافية OnPlatform blocks

```
<VerticalStackLayout>
    <VerticalStackLayout.Padding>
       <OnPlatform x:TypeArguments="Thickness">
         <On Platform="iOS" Value="30,60,30,30" />
            <On Platform="Android" Value="30" />
            <On Platform="WinUI" Value="30" />
       </OnPlatform>
    </VerticalStackLayout.Padding>
    ...
</VerticalStackLayout>
```

يمكنك تطبيق هذه التقنية نفسها على خصائص أخرى. يغير المثال التالي لون خلفية التخطيط على صفحة stack layout's background color إلى Silver على iOS إلى أخضر على Android إلى أصفر على Windows

```
<VerticalStackLayout>
    ...
    <VerticalStackLayout.BackgroundColor>
        <OnPlatform x:TypeArguments="Color">
           <On Platform="iOS" Value="Silver" />
           <On Platform="Android" Value="Green" />
           <On Platform="WinUI" Value="Yellow" />
        </OnPlatform>
    </VerticalStackLayout.BackgroundColor>
    ...
</VerticalStackLayout>
```

بناء الجملة هذا طويل قليلاً، يوجد بناء جملة مختصر، متاح للملحق OnPlatform يمكنك تبسيط المثال الذي يعين الحشو sets the padding كما يلي:

```
<VerticalStackLayout Padding="{OnPlatform
iOS='30,60,30,30', Default='30'}">
    <!--XAML for other controls goes here -->
</VerticalStackLayout>
```

يمكنك تعين قيمة افتراضية لخاصية، مع أي قيم خاصة بالنظام، في هذا النموذج، يتم استنتاج معلمة النوع type parameter من الخاصية التي يتم تطبيق السمة `OnPlatform attribute` عليها.

لتعيين لون الخلفية background color يمكنك استخدام جزء XAML هذا بدلاً من المثال الآخر، السابق:

```
<VerticalStackLayout BackgroundColor="{OnPlatform
WinUI=Yellow, iOS=Silver, Android=Green}">
    ...
</VerticalStackLayout>
```

اختبار بسيط:

١- ما ملحق العلامات markup extension الذي يمكنك استخدامه في XAML للكشف عن النظام الأساسي الذي يعمل عليه التطبيق؟

٢- ما هو الغرض من السمة `TypeArguments attribute` للملحق `OnPlatform extension`

حل الاختبار:

١- OnPlatform

٢- يحدد معلمة النوع type parameter للملحق، ويضمن إنشاء النوع الصحيح من الملحق.

٨ تمرين: إضافة سلوك behavior إلى صفحة XAML

قمت مسبقاً بتعديل تطبيق الملاحظات لنقل تخطيط واجهة المستخدم من تعليمات C# إلى XAML أنت الآن جاهز لإضافة الميزات التالية إلى الصفحة:

- دعم تخصيص لون خط ولون الخلفية لملصق label's font color and background color وعناصر التحكم buttons, editor control بهذه الطريقة، يكون من السهل ضبط التطبيق لجعله أكثر سهولة للمستخدمين الذين يحتاجون إلى واجهة مستخدم عالية التباين.
- ضبط ارتفاع عنصر editor على نظامي التشغيل Android وiOS عند التشغيل على Windows يكون عرض عنصر التحكم هذا كاف للسماح للمستخدم بإدخال قدر معقول من النص قبل التمرير، على هاتف Android أو iPhone يؤدي العرض الأضيق إلى حدوث التمرير بسرعة أكبر، لذلك من المفيد توفير مساحة عمودية أكبر more vertical space

استخدام مورد ثابت في XAML

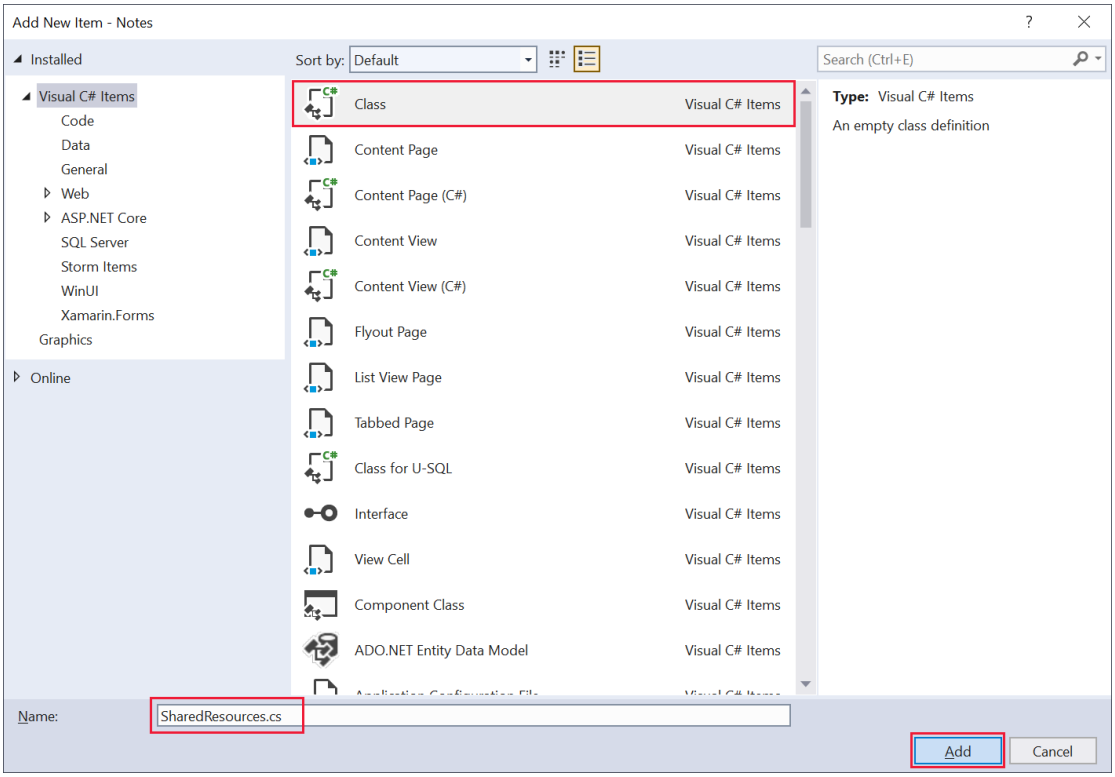
ستقوم بإنشاء فئة ثابتة static class للاحتفاظ بلون خط التطبيق وقيم لون الخلفية font color and background color values ثم ستستخدم ملحق العلامات `x:Static markup extension` لقراءة هذه القيم من الفئة وتطبيقها على علامات XAML markup لعناصر التحكم على الصفحة.

١- في Visual Studio ارجع إلى تطبيق الملاحظات Notes الذي قمت بتحريره في التمرين السابق

ملحوظة: تتوفر نسخة عمل من التطبيق في مجلد exercise2 في مستودع التمرين الذي نسخته في بداية التمرين السابق.

٢- في نافذة مستكشف الحلول Solution Explorer انقر بزر الماوس الأيمن فوق مشروع الملاحظات Notes وحدد Add ثم حدد Class

٣- في مربع الحوار Add New Item تأكد من تحديد قالب Class قم بتسمية ملف الفئة الجديد `SharedResources.cs` ثم حدد Add



٤- في ملف SharedResources.cs استبدل التوجيهات `using` directives بالتعليمات البرمجية التالية، وضع علامة على SharedResources class ك static

```
namespace Notes;

static class SharedResources
{

}
```

٥- أضف حقل القراءة فقط الثابت `static readonly field` **`FontColor`** إلى فئة SharedResources. يوفر هذا الحقل حاليًا قيمة تتوافق مع اللون الأزرق، ولكن يمكنك تعديله باستخدام أي مجموعة صالحة من قيم RGB

أضف الحقل FontColor static readonly إلى SharedResources class يوفر هذا الحقل حالياً قيمة تتوافق مع اللون الأزرق، ولكن يمكنك تعديله باستخدام أي تركيبة صالحة من قيم RGB

```
static class SharedResources
{
```

```
    public static readonly Color FontColor =
Color.FromRgb(0, 0, 0xFF);
}
```

٦- أضف حقلاً ثانياً static readonly باسم BackgroundColor وقم بتعيينه إلى لون من اختيارك:

```
static class SharedResources
{
    ...
    public static readonly Color BackgroundColor =
Color.FromRgb(0xFF, 0xF0, 0xAD);
}
```

٧- أفتح الملف MainPage.xaml

٨- أضف تعريف مساحة الاسم التالي ل XML إلى العنصر ContentPage element قبل السمة x:Class attribute يجلب هذا الإعلان classes in the C# Notes namespace إلى النطاق في صفحة XAML

```
<ContentPage ...
      xmlns:notes="clr-namespace:Notes"
      x:Class="Notes.MainPage"
      ...>
```

٩- أضف سمة TextColor الموضحة في التعليمات التالية إلى عنصر التحكم Label تستخدم هذه العلامات x:Static markup extension لاسترداد القيم المخزنة في الحقول الثابتة values stored in the static fields في فئة SharedResources class

```
<Label Text="Notes"
    HorizontalOptions="Center"
    FontAttributes="Bold"
    TextColor="{x:Static
Member=notes:SharedResources.FontColor}" />
```

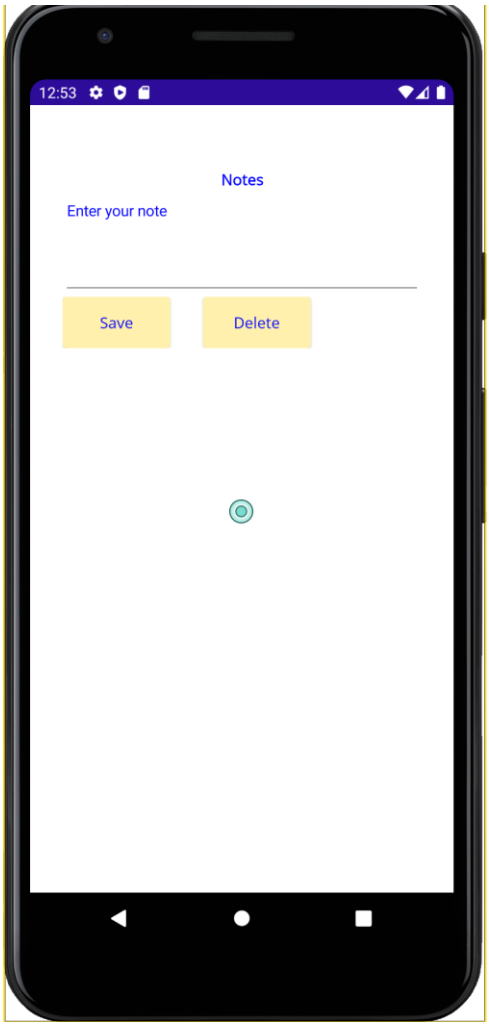
١٠- استخدم ملحق العلامات x:Static mark-up extension لتعيين سمات TextColor and BackgroundColor attributes لعناصر Editor, Button controls يجب أن يبدو الترميز المكتمل لملف MainPage.xaml الخاص بك على النحو التالي:

```
<ContentPage
xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:notes="clr-namespace:Notes"
    x:Class="Notes.MainPage">

 <VerticalStackLayout Padding="30,60,30,30">
     <Label Text="Notes"
         HorizontalOptions="Center"
         FontAttributes="Bold"
         TextColor="{x:Static
Member=notes:SharedResources.FontColor}" />

     <Editor x:Name="editor"
             Placeholder="Enter your note"
             HeightRequest="100"
             TextColor="{x:Static
Member=notes:SharedResources.FontColor}"/>

   <Grid Grid.Row="2" ColumnDefinitions="Auto,30,Auto">

       <Button Grid.Column="0"
               Text="Save"
               WidthRequest="100"
               TextColor="{x:Static
Member=notes:SharedResources.FontColor}"
               BackgroundColor="{x:Static
Member=notes:SharedResources.BackgroundColor}"
               Clicked="OnSaveButtonClicked" />

       <Button Grid.Column="2"
               Text="Delete"
               WidthRequest="100"
               TextColor="{x:Static
Member=notes:SharedResources.FontColor}"
               BackgroundColor="{x:Static
Member=notes:SharedResources.BackgroundColor}"
               Clicked="OnDeleteButtonClicked" />
   </Grid>
    </VerticalStackLayout>

</ContentPage>
```

ملاحظة:

تحتوي تعليمات XAML هذه على تكرار العلامات the markup التي تعين خصائص TextColor and BackgroundColor يتيح لك XAML تحديد الموارد resources التي يمكن تطبيقها globally عبر تطبيق، باستخدام قاموس موارد resource dictionary في ملف App.xaml سنصف هذه التقنية في وحدة لاحقة.

١١- أعد إنشاء التطبيق وقم بتشغيله باستخدام Windows تحقق من أن الألوان تطابق تلك التي حددتها في فئة SharedResources إذا كان لديك الوقت، فحاول أيضاً تشغيل التطبيق باستخدام محاكي Android



١٢- ارجع إلى Visual Studio عند الانتهاء.

إضافة تخصيص خاص بنظام التشغيل Add platform-specific customization

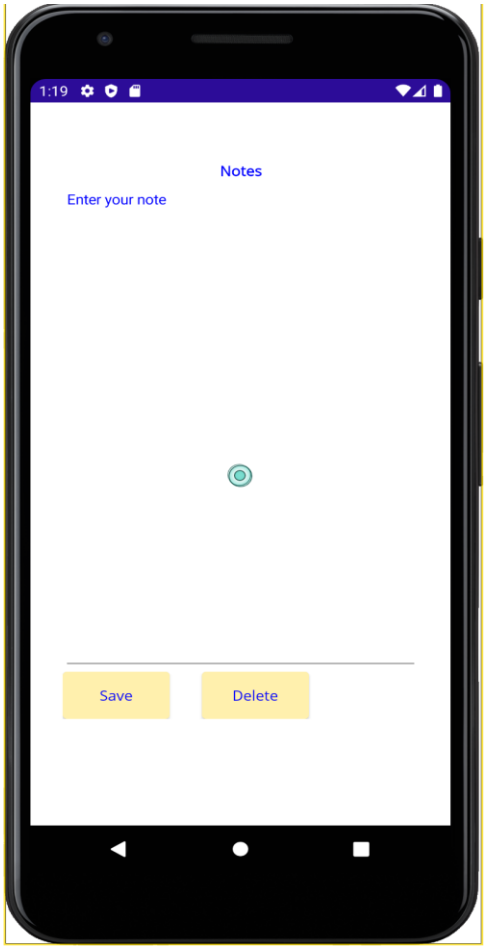
١- أفتح ملف MainPage.xaml في Visual Studio

٢- حدد تعريف Editor ثم قم بتعديل قيمة الخاصية HeightRequest كما هو موضح في المثال التالي:

```
<Editor x:Name="editor"
    ...
    HeightRequest="{OnPlatform 100, Android=500,
iOS=500}"
    .../>
```

تعين هذه العلامات ارتفاع عنصر التحكم الافتراضي إلى 100 وحدة، ولكنها تزيده إلى 500 على Android

٣- أعد إنشاء التطبيق وقم بتشغيله باستخدام Windows ثم Android يجب أن يبدو التطبيق كما يلي على كل نظام:



٩ الملخص

واجهة المستخدم ذات تعليمة برمجية coded UI تجعل من الصعب إدارة التخطيط والسلوك، غالبًا ما يتضمن هذا النهج التخطيط والمنطق السلوكي، ويؤدي إلى اقتران ضيق بين الاثنين، يمكن أن يتسبب التغيير في تصميم واجهة المستخدم في حدوث تأثير غير مباشر على بقية قاعدة التعليمات، ربما يكون الحفاظ على قاعدة تعليمات برمجية لا يوجد فيها فصل واضح بين واجهة المستخدم والسلوك أمرًا صعبًا.

يتيح لك .NET MAUI. تعريف واجهة مستخدم باستخدام XAML يتيح لك هذا الفصل التركيز على منطق السلوك في ملفات التعليمات البرمجية C# يمكن لمصممي واجهة المستخدم الآن التركيز على واجهة المستخدم، بينما يمكن للمبرمجين التركيز على التعليمات البرمجية.

يتيح لك .NET MAUI XAML. بتخصيص واجهة المستخدم لكل نظام تشغيل باستخدام `OnPlatform markup extension` يتيح لك هذا النهج تصميم التطبيقات التي يمكنها الاستفادة من ميزات واجهة المستخدم الخاصة بنظام التشغيل، ولكن لا تزال تبدو جيدة على جميع الأنظمة الأخرى.

في هذه الوحدة، تعلمت كيفية استخدام XAML لأفضل تأثير لتصميم واجهة المستخدم لتطبيق متعدد الأنظمة، على وجه التحديد، تعلمت:

- فوائد استخدام XAML على تعريف واجهة المستخدم لتطبيق .NET MAUI في C#
- كيفية إنشاء الصفحات pages وعناصر التحكم controls وتعيين خصائصها set their properties باستخدام XAML
- كيفية التعامل مع أحداث واجهة المستخدم handle UI events وربطها في XAML
- كيفية إنشاء واستخدام ملحقات علامات XAML markup extensions
- كيفية تعيين قيم خاصة بنظام التشغيل set platform-specific values في علامات XAML markup

الوحدة الثالثة

تخصيص التخطيط layout في صفحات .NET MAUI XAML

إنشاء واجهات مستخدم متناسقة عبر أجهزة مختلفة باستخدام StackLayout and Grid في صفحات واجهة مستخدم التطبيق متعدد الأنظمة Extensible Application Markup Language (XAML) pages الخاصة بـ .NET Multi-platform App UI (MAUI)

الأهداف التعليمية

خلال هذه الوحدة، سوف تتمكن مما يلي:

- ترتيب عناصر واجهة المستخدم user interface elements في تطبيق وتحديد حجمها size
- عرض طرق العرض views في قائمة رأسية أو أفقية vertical or horizontal باستخدام عنصر التخطيط StackLayout
- عرض طرق العرض views في الصفوف والأعمدة rows and columns باستخدام التخطيط الشبكي Grid

محتويات الوحدة:

١- مقدمة

٢- تحديد حجم طريقة العرض Specify the size of a view

٣- تمرين - استكشاف خيارات المحاذاة alignment options

٤- ترتيب طرق العرض views باستخدام StackLayout

٥- تمرين - استخدام StackLayout لإنشاء واجهة مستخدم build a user interface

٦- ترتيب طرق العرض views باستخدام الشبكة Grid

٧- تمرين - استخدام Grid لإنشاء واجهة مستخدم user interface

٨- الملخص

١ المقدمة

تساعدك لوحات تخطيط .NET MAUI layout panels على إنشاء واجهات مستخدم ملائمة لتطبيقك عبر مجموعة واسعة من الأجهزة.

لنفترض أنك تقوم بإنشاء تطبيق حاسبة تلميح، تخطط لتوزيعه على العديد من أجهزة الكمبيوتر والأجهزة الأخرى، يمكن أن يكون لكل جهاز حجم شاشة وكثافة بكسل مختلفة، هدفك هو جعل التطبيق يبدو مشابهًا قدر الإمكان على جميع الأجهزة، ترغب في تجنب الحساب اليدوي لحجم العرض وموضعه لكل حجم شاشة، يتضمن .NET MAUI نظام إدارة تخطيط يقوم بهذه العمليات الحسابية تلقائيًا نيابة عنك، يمكنك وضع طرق العرض views داخل لوحات التخطيط layout panels التي تدير تلقائيًا حجم وموضع طرق العرض التابعة لها child views تلقائيًا، تسهل اللوحات إنشاء واجهات مستخدم متناسقة عبر أجهزة مختلفة.

في هذه الوحدة، يمكنك إنشاء واجهة مستخدم لتطبيق متعدد المنصات .NET Multi-platform App UI (MAUI) يبدو متشابهًا عبر أجهزة مختلفة، تبدأ بتعيين الحجم والموضع المفضلين للعرض، ثم تقوم بترتيب طرق العرض عموديًا باستخدام StackLayout بعد ذلك، تقوم بوضع طرق العرض في صفوف وأعمدة باستخدام Grid بحلول نهاية الوحدة، سيكون لديك تطبيق .NET MAUI. ملائم على كل أنوع الأجهزة، وحجم الشاشة.

٢ تحديد حجم طريقة العرض Specify the size of a view

من الصعب تصميم واجهة مستخدم متناسقة عبر أجهزة متعددة، لأن أحجام الأجهزة مختلفة، ولها كثافة بكسل مختلفة، فكر في الأجهزة المتوفرة: الأجهزة المحمولة، والكمبيوتر اللوحي، وسطح المكتب وما إلى ذلك. كيف يمكننا إنشاء واجهة مستخدم تبدو متشابهة في كل منها؟

يوفر .NET Multi-platform App UI (MAUI) لوحات تخطيط layout panels لمساعدتك في إنشاء واجهات مستخدم متسقة، لوحة التخطيط مسؤولة عن تغيير حجم طرق عرض العناصر التابعة لها views of its children وتحديد موضعها، في هذا الدرس، ستتعرف على كيفية عمل نظام التخطيط في .NET MAUI بشكل خاص، ننظر في كيفية تغيير حجم طرق العرض بشكل افتراضي، وكيفية طلب حجم وموضع معينين لطريقة عرض في وقت التشغيل.

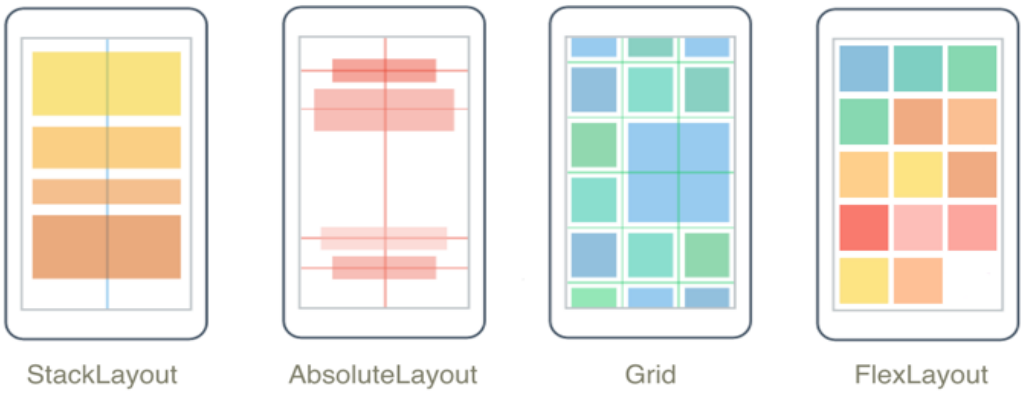
ما هي لوحة التخطيط؟ layout panel

لوحة التخطيط هي حاوية .NET MAUI. تحتوي على مجموعة من طرق العرض التابعة child views وتحدد حجمها وموضعها size and position تقوم لوحات التخطيط بإعادة الحساب تلقائيًا عند تغير حجم التطبيق؛ على سبيل المثال، عندما يقوم المستخدم بتدوير الجهاز.

ملاحظة:

يشير مصطلح طرق العرض view أو طرق العرض الفرعية child view إلى عنصر تحكم control موضوع على لوحة تخطيط، يمكن أن تكون طريقة العرض تسمية أو زر أو حقل إدخال label, button, entry field أو أي نوع آخر من العناصر المرئية التي يدعمها .NET MAUI.

يحتوي .NET MAUI. على لوحات تخطيط متعددة يمكنك الاختيار من بينها، كل لوحة تدير طرق العرض التابعة الخاصة بها بشكل مختلف، يوضح الرسم التوضيحي التالي نظرة عامة تصورية لبعض الخيارات الأكثر شيوعاً.



StackLayout يرتب عناصره الفرعية في صف أو عمود واحد، بالإضافة إلى هذا التخطيط هناك أيضًا VerticalStackLayout and HorizontalStackLayout محسّنان عندما لا تحتاج إلى تغيير الاتجاه.

AbsoluteLayout يقوم بترتيب طرق العرض الفرعية باستخدام إحداثيات x و y

Grid تقوم بترتيب طرق العرض الفرعية في خلايا يتم إنشاؤها من تقاطع الصفوف والأعمدة rows and columns

FlexLayout يقوم بترتيب عروضه الفرعية مثل StackLayout باستثناء أنه يمكنها الالتفاف إذا لم يحتويها صف أو عمود واحد.

ملاحظة:

هناك أيضاً نوع خامس من لوحة التخطيط يسمى RelativeLayout يمكنك من تحديد كيفية ترتيب طرق العرض التابعة بالنسبة لبعضها البعض. يجب استخدام FlexLayout بدلاً من RelativeLayout لأن لوحة التخطيط FlexLayout تعمل بشكل أفضل. RelativeLayout مضمن في .NET MAUI للتوافق مع الإصدارات السابقة مع تطبيقات Xamarin القديمة، ولتسهيل الانتقال إلى .NET MAUI

العملية النموذجية لإنشاء صفحة .NET MAUI هي إنشاء لوحة تخطيط ثم إضافة طرق عرض تابعة إليها، عند إضافة طرق عرض إلى تخطيط، يمكنك التأثير على حجمها وموضعها، ومع ذلك، فإن اللوحة لها القول الفصل استنادًا إلى خوارزميات التخطيط الداخلي.

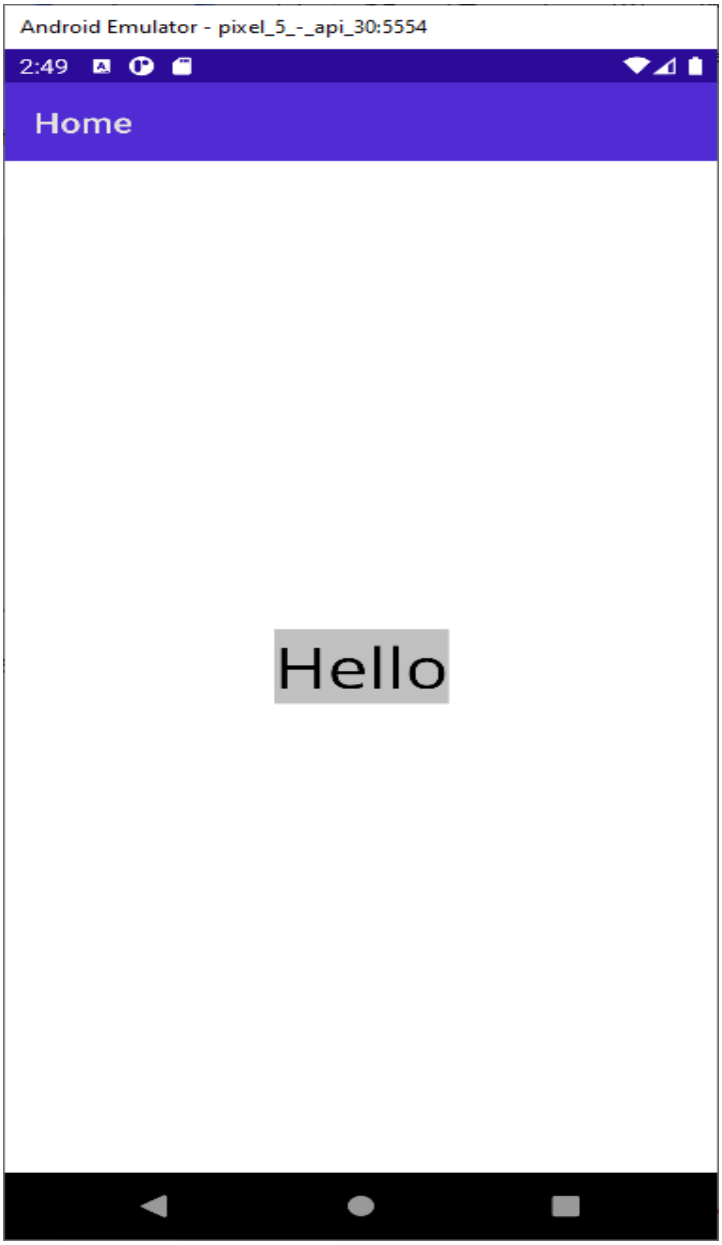
قبل أن تنظر إلى كيفية طلب حجم معين لطريقة عرض، دعنا نرى كيف يقوم نظام التخطيط بتعيين أحجام طرق العرض بشكل افتراضي.

الحجم الافتراضي لعرض ما Default size of a view

إذا لم تحدد حجم طريقة العرض، فسيزداد حجمها تلقائياً بحيث تكون كبيرة بما يكفي لتلائم محتواها، على سبيل المثال، ضع في اعتبارك لغة علامات (XAML):

```
<Label
    Text="Hello"
    BackgroundColor="Silver"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    FontSize="40"/>
```

يعرف هذا المثال Label لعرض الكلمة Hello على خلفية فضية، نظراً لأنك لا تحدد الحجم، يتم تلقائياً تغيير حجم الملصق ليلائم الكلمة Hello تظهر الصورة التالية الملصق معروض على جهاز Android



ملاحظة:

يمكنك تعيين لون خلفية الملصق لمساعدتك في تحديد حجمه وقت التشغيل، هذه تقنية تصحيح أخطاء جيدة يجب وضعها في الاعتبار أثناء إنشاء واجهة المستخدم الخاصة بك.

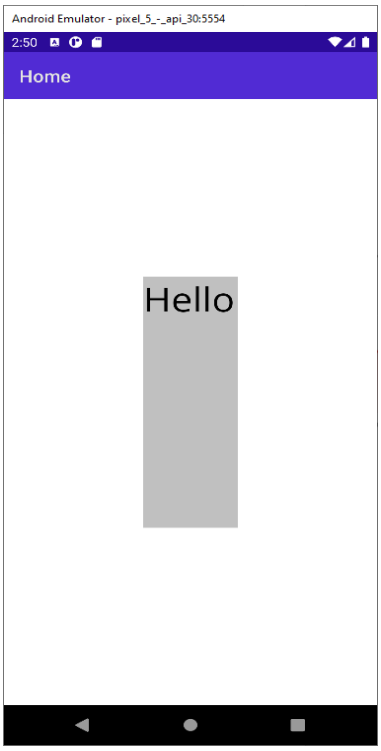
تحديد حجم طريقة العرض Specify the size of a view

عند إنشاء واجهة مستخدم، من الشائع أن ترغب في التحكم في حجم طريقة العرض، على سبيل المثال، تخيل أنك تقوم بإنشاء صفحة تسجيل دخول وتريد أن يكون زر تسجيل الدخول نصف عرض الشاشة تماماً، إذا استخدمت الحجم الافتراضي لعرض ما، فسيكون الزر على قدر حجم النص Text "تسجيل الدخول" هذا الحجم ليس كبيراً بما فيه الكفاية، لذلك تحتاج إلى تحديد الحجم بنفسك.

تحدد الفئة الأساسية `View base class` خاصيتين تؤثران على حجم العرض `WidthRequest and HeightRequest` يتيح لك `WidthRequest` تحديد العرض، ويتيح لك `HeightRequest` تحديد الارتفاع. كلتا الخاصيتين من النوع رقمي عشري double

فيما يلي مثال يوضح كيفية تحديد عرض الملصق وارتفاعه في XAML:

```
<Label
    Text="Hello"
    BackgroundColor="Silver"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    WidthRequest="100"
    HeightRequest="300"
    FontSize="40"/>
```



ملاحظة:

لا يزال Label موجود في الوسط، على الرغم من أن text ليس في وسط الملصق center of the label

أحد الأشياء الجديرة بالملاحظة هو أسماء هذه الخصائص، تحتوي كلتا الخاصيتين على كلمة **request** تعني هذه الكلمة أن لوحة التخطيط قد لا تحترمها في وقت التشغيل. تقرأ لوحة التخطيط هذه القيم أثناء حسابات الحجم، وتحاول تلبية الطلبات إذا أمكن ذلك. إذا لم تكن هناك مساحة كافية، يُسمح للوحة التخطيط بتجاهل القيم.

وحدات الحجم Size units

عند تعيين WidthRequest و HeightRequest يمكنك استخدام القيم الحرفية literal values مثل 100 على مستوى .NET MAUI. لا تحتوي هذه القيم على وحدات units فهي ليست نقاطًا أو بكسلات points or pixels هي مجرد قيم من النوع double يقوم .NET MAUI بتمرير هذه القيم إلى نظام التشغيل في وقت التشغيل.

إنه نظام التشغيل الذي يوفر السياق المطلوب لتحديد ما تعنيه الأرقام:

- على نظام iOS تسمى القيم نقاطاً the values are called points
- على نظام Android فهي عبارة عن بكسلات مستقلة عن الكثافة -density independent pixels

عرض حجم العرض Rendered size of a view

نظرًا لأن الأمر متروك للوحة التخطيط لتحديد حجم العرض، فلا يمكنك استخدام WidthRequest وHeightRequest لإخبارك بالحجم الفعلي في وقت التشغيل. على سبيل المثال، تخيل أنك قمت بتعيين WidthRequest على 100 لـ Label ولكن اللوحة لا تحتوي على مساحة كافية لتلبية الطلب، بدلاً من ذلك، تمنح اللوحة علامتك عرضًا يبلغ 80 عند هذه المرحلة، إذا قمت بفحص قيمة خاصية WidthRequest فستقول 80 على الرغم من أن القيمة المعروضة هي 80

لحل هذه المشكلة، توفر الفئة الأساسية View base class خاصيتين أخريين تسمى Width و Height هذه الخصائص هي من نوع double وتمثلان عرض وارتفاعه View استخدم الخاصيتين Width و Height كلما قمت باسترداد حجم طريقة العرض.

تحديد موضع طريقة عرض Specify the position of a view

تحتاج أيضًا إلى تعيين موضع طريقة العرض، على سبيل المثال، تذكر أنك في مثال صفحة تسجيل الدخول أردت تغيير حجم زر تسجيل الدخول ليكون نصف عرض الشاشة، نظراً لأن زر تسجيل الدخول ليس العرض الكامل للشاشة، فهناك بعض المساحة المتوفرة لتحريكه، يمكنك وضعه على الجانب الأيسر أو على الجانب الأيمن أو في وسط الشاشة.

تحتوي الفئة الأساسية View base class على خاصيتين تستخدمهما لتعيين موضع العرض VerticalOptions and HorizontalOptions تؤثر هذه الإعدادات على كيفية وضع طريقة العرض داخل المستطيل المخصص لها من قبل لوحة التخطيط، يمكنك تحديد أنك تريد محاذاة طريقة العرض مع أحد حواف المستطيل الأربعة، أو أنك تريد أن يشغل المستطيل بالكامل.

تحديد قيمة VerticalOptions أو HorizontalOptions أكثر صعوبة من تعيين الحجم لأنها من النوع `LayoutOptions`

ما هو نوع LayoutOptions؟

LayoutOptions هو نوع C# يغلف تفضيلين للتخطيط Alignment وExpands كلتا الخاصيتين مرتبطتان بالموضع، لكنهما غير مرتبطتين ببعضهما البعض. فيما يلي شكل تعريف النوع:

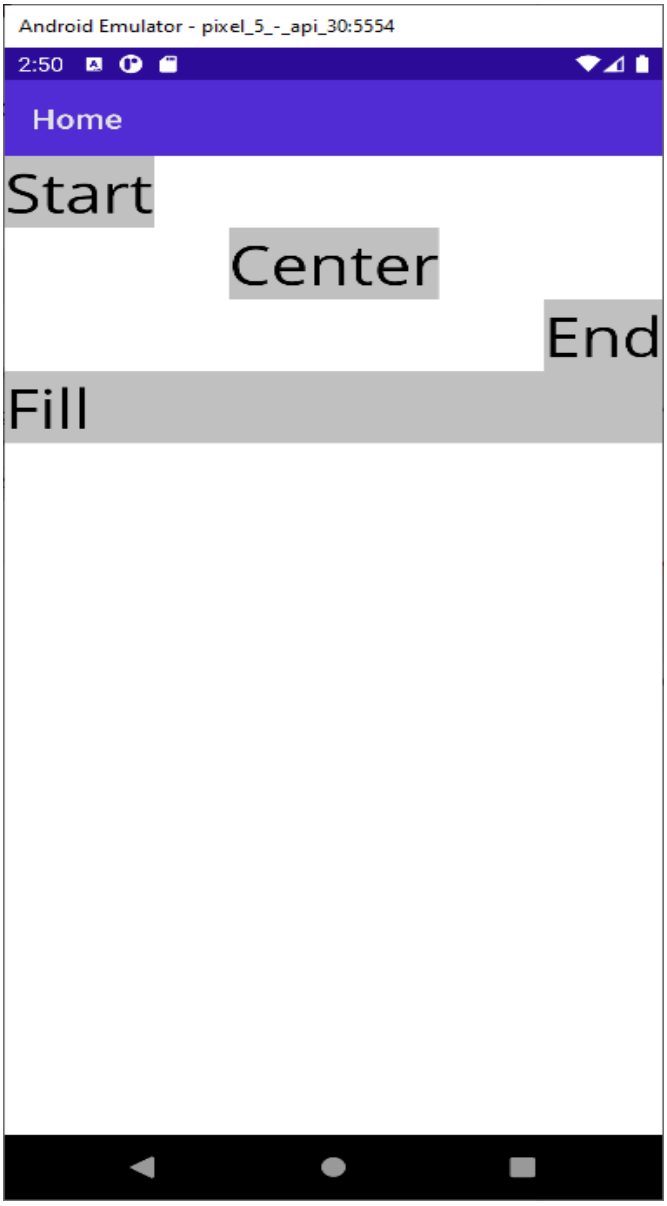
```
public struct LayoutOptions
{
    public LayoutAlignment Alignment { get; set; }
    public bool Expands { get; set; }
    ...
}
```

بعد ذلك، سننظر عن كثب Alignment "المحاذاة أو التنسيق" لأنه خيار التخطيط الأكثر شيوعاً وبديهية.

ما هو تعداد LayoutAlignment enumeration

LayoutAlignment هو تعداد يحتوي على أربع قيم `Start, Center, End, Fill` يمكنك استخدام هذه القيم للتحكم في كيفية وضع العرض التابع داخل المستطيل المعطى له، من قبل لوحة التخطيط الخاصة به. على سبيل المثال، ضع في اعتبارك التعليمات البرمجية التالية، ولقطة شاشة Android

```
<StackLayout>
    <Label Text="Start"
HorizontalOptions="Start"
BackgroundColor="Silver" FontSize="40" />
    <Label Text="Center"
HorizontalOptions="Center"
BackgroundColor="Silver"  FontSize="40" />
    <Label Text="End" HorizontalOptions="End"
BackgroundColor="Silver"  FontSize="40"/>
    <Label Text="Fill" HorizontalOptions="Fill"
BackgroundColor="Silver"  FontSize="40"/>
</StackLayout>
```



يستخدم المثال تخطيطًا عموديًا vertical StackLayout بحيث يتم منح كل طريقة عرض فرعية child view صفًا. تحدد HorizontalOptions موضع طريقة العرض داخل صفها.

ما هو التوسيع Expands

الخاصية الثانية لبنية LayoutOptions هي Expands الخاصية Expands هي قيمة منطقية bool تسمح في Xamarin.Forms لطريقة عرض في StackLayout بأخذ مساحة إضافية إذا كانت متوفرة، هذه الخاصية أصبحت قديمة الآن ولم تعد مستخدمة في NET MAUI. لاحقًا، سنستكشف كيفية تحقيق نفس نوع التوسيع في درس تخطيط الشبكة Grid layout

أبحث عن إجابة السؤال التالي:

- ماذا يفعل LayoutPanel

٣ تمرين - استكشاف خيارات المحاذاة alignment options

في هذا التمرين، يمكنك استخدام تطبيق .NET Multi-platform App UI (MAUI) لمشاهدة تأثير خيارات التخطيط الأساسية الأربعة، عند تطبيقها على طريقة عرض مضمنة في Grid لن تكتب التعليمات البرمجية في التمرين، بدلاً من ذلك، يمكنك استخدام الحل المتوفر، وتحديد الأزرار لتغيير خيارات تخطيط Label

تستخدم هذا الدرس .NET 9.0 SDK. تأكد من تثبيت .NET 9.0 عن طريق تشغيل الأمر التالي في موجة الأوامر command terminal

```
dotnet --list-sdks
```

يظهر إخراج مشابه للتالي:

```
8.0.100 [C:\Program Files\dotnet\sdk]
9.0.100 [C:\Program Files\dotnet\sdk]
```

تأكد من إدراج إصدار يبدأ بـ 9 إذا لم يتم سرد أي منها أو لم يتم العثور على الأمر، فقم بتثبيت .NET 9.0 SDK.

أفتح the starter solution

١- انسخ أو نزل exercise repo من GitHub

ملاحظة:

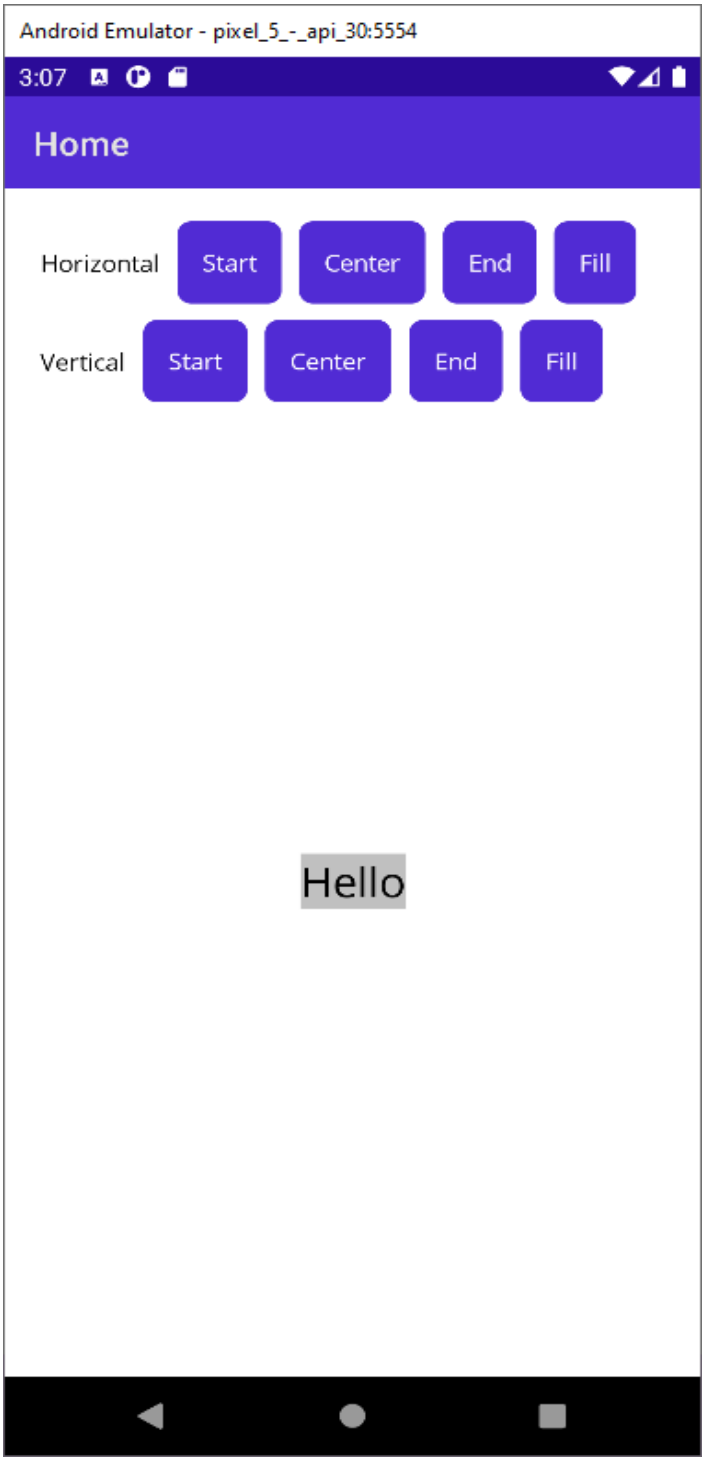
من الأفضل نسخ أو تنزيل محتوى التمرين إلى مسار مجلد قصير، مثل C:\dev لتجنب تجاوز الملفات المنشأة، الحد الأقصى لطول المسار.

٢- افتح the starter solution من المجلد exercise1/Alignment باستخدام Visual Studio أو نفس المجلد في Visual Studio Code

اختبار سلوك التطبيق Test the application behavior

قم بتشغيل التطبيق لاختبار خيارات التخطيط LayoutOptions والاطلاع على كيفية تغيير خيارات التخطيط المختلفة لحجم الملصق وموضعه size and position of the label

١- اختبر التطبيق عن طريق التفاعل مع الأزرار التي تغيّر خيارات التخطيط الأفقي horizontal والعمودي vertical حددها ولاحظ ما يحدث، تظهر الصورة التالية ما يحدث إذا قمت بتحديد توسيط Center لكل من خيارات المحاذاة alignment options الأفقية والعمودية:



يمكن لخيارات المحاذاة (`Start`, `Center`, `End`, `Fill`) تغيير كل من حجم ومحاذاة العرض size and the alignment of a view

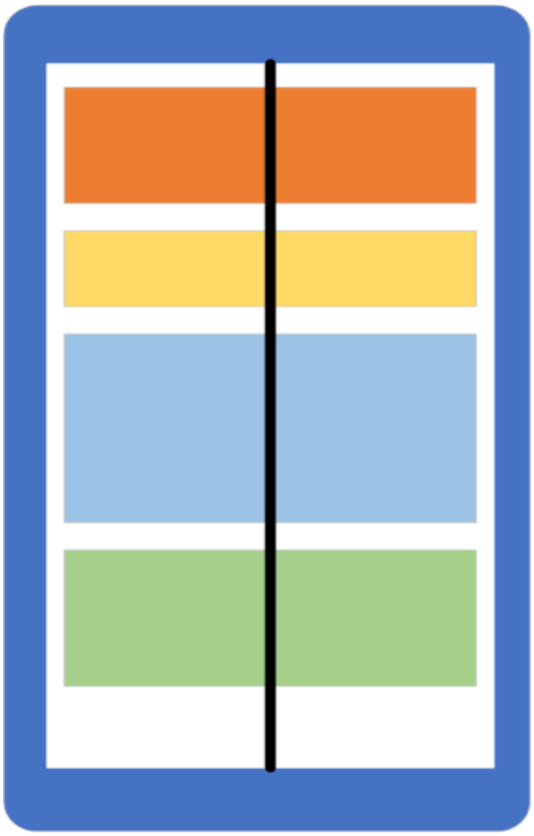
٤ ترتيب طرق العرض views باستخدام StackLayout

يعد تجميع طرق العرض في قائمة رأسية أو أفقية تصميمًا شائعًا لواجهات المستخدم، فكر في بعض الصفحات الشائعة في التطبيقات، تتضمن الأمثلة صفحات تسجيل الدخول والتسجيل والإعدادات، تحتوي كافة هذه الصفحات عادةً على محتوى مكدس أو مجمع. في هذا الدرس، ستتعلم كيفية ترتيب طرق العرض في قائمة عمودية أو أفقية باستخدام `StackLayout` وعناصر التحكم المحسّنة `VerticalStackLayout` and `HorizontalStackLayout`

ما هي StackLayout, VerticalStackLayout, HorizontalStackLayout

`StackLayout` هي حاوية تخطيط تنظم طرق العرض الفرعية child views التابعة لها، من اليسار إلى اليمين أو من أعلى إلى أسفل left-to-right or top-to-bottom يعتمد الاتجاه على الخاصية Orientation الخاصة بها، القيمة الافتراضية هي من الأعلى إلى الأسفل top-to-bottom

يوضح الرسم التوضيحي التالي تخطيط تصوري لـ vertical StackLayout



- `StackLayout` يحتوي على قائمة من العناصر الفرعية `Children` التي يرثها من فئته الأساسية `Layout<T>` base class تخزن القائمة طرق

العرض views وهو أمر مفيد لأن معظم عناصر واجهة المستخدم UI elements التي تعمل بها في واجهة مستخدم التطبيقات متعددة المنصات .NET. Multi-platform App UI (MAUI) مشتقة من View تشتق لوحات التخطيط Layout panels أيضًا من View مما يعني أنه يسمح بتداخل اللوحات إذا كنت بحاجة إلى ذلك.

- تعد VerticalStackLayout and HorizontalStackLayout من التخطيطات المفضلة للاستخدام، عندما تعلم أن اتجاهك orientation لن يتغير، لأنها مُحسّنة للأداء.

كيفية إضافة طرق عرض add views إلى StackLayout

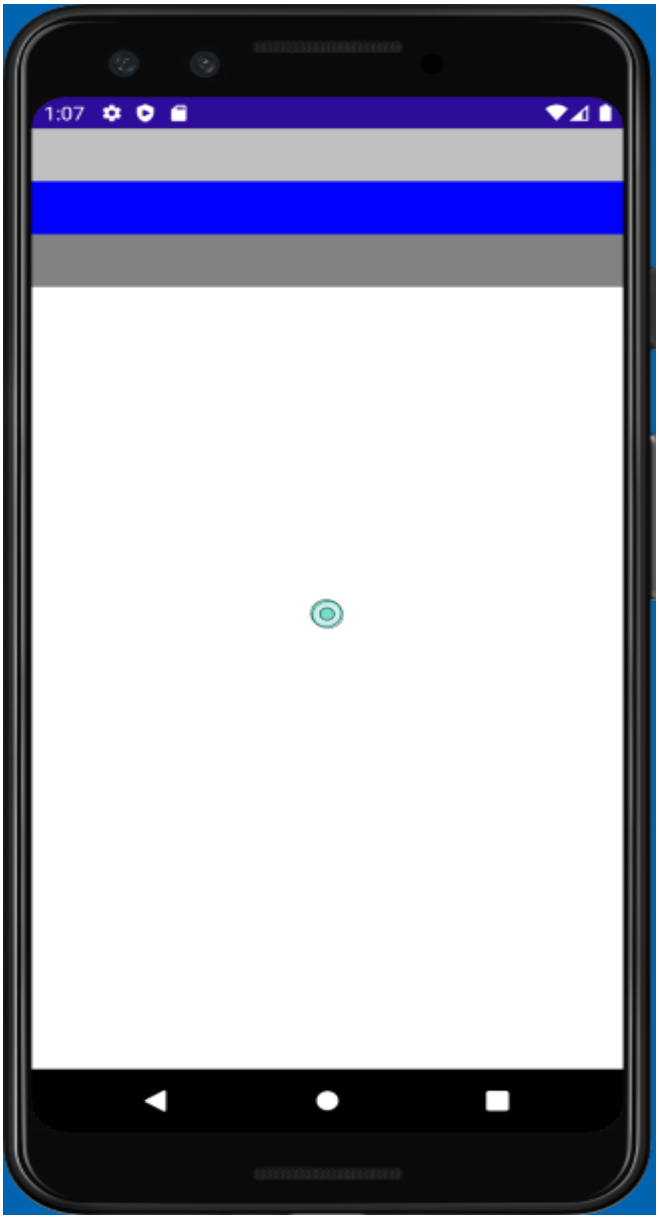
في .NET MAUI. يمكنك إضافة طرق عرض إلى StackLayout في التعليمات البرمجية C# أو في Extensible Application Markup Language (XAML) فيما يلي مثال على ثلاث طرق عرض تمت إضافتها باستخدام التعليمات البرمجية:

```
<StackLayout x:Name="stack">
</StackLayout>
```

C# code

```
public partial class MainPage : ContentPage
{
   public MainPage()
   {
      InitializeComponent();

var a = new BoxView { Color = Colors.Silver, HeightRequest
= 40 };
var b = new BoxView { Color = Colors.Blue, HeightRequest =
40 };
var c = new BoxView { Color = Colors.Gray, HeightRequest =
40 };

      stack.Children.Add(a);
      stack.Children.Add(b);
      stack.Children.Add(c);
   }
}
```

يمكنك إضافة طرق العرض إلى مجموعة Children وسيقوم StackLayout تلقائيًا بوضع طرق العرض في قائمة عمودية vertical list إليك ما يبدو عليه على Android



للقيام بنفس الشيء في XAML قم بتضمين العناصر الفرعية Children داخل علامات StackLayout يضيف محلل XAML العناصر الفرعية المضمنة إلى مجموعة العناصر الفرعية Children تلقائيًا، لأن العناصر الفرعية هي ContentProperty لجميع لوحات التخطيط.

فيما يلي مثال XAML على نفس طرق العرض الثلاث التي تمت إضافتها إلى StackLayout

```
<StackLayout>
    <BoxView Color="Silver" HeightRequest="40" />
    <BoxView Color="Blue" HeightRequest="40" />
    <BoxView Color="Gray" HeightRequest="40" />
</StackLayout>
```

كيفية ترتيب طرق العرض views في StackLayout

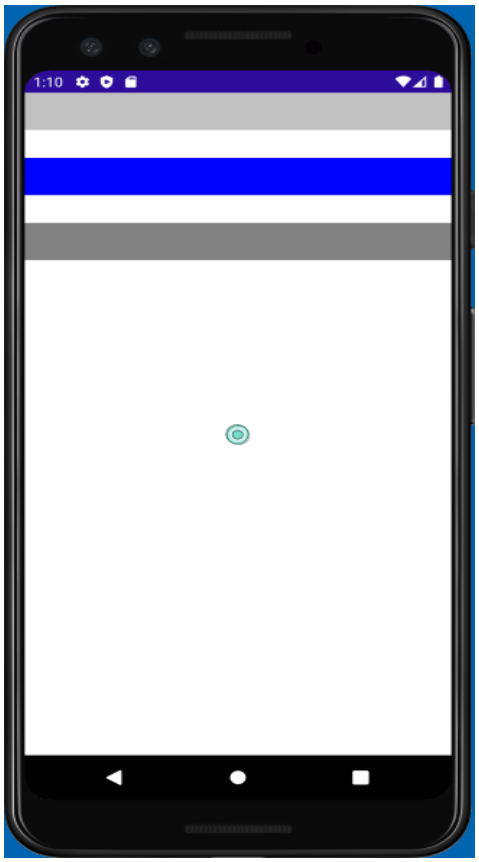
يحدد ترتيب views في مجموعة العناصر الفرعية Children ترتيب تخطيطها عند عرضها، بالنسبة للعناصر views المضافة في XAML يتم استخدام الترتيب النصي، بالنسبة للعناصر الفرعية Children المضافة في التعليمات، يحدد الترتيب الذي يستدعي أسلوب الإضافة `Add method` ترتيب التخطيط.

كيفية تغيير المسافة space بين طرق العرض في StackLayout

من الشائع أن ترغب في وجود مسافة بين العناصر الفرعية في StackLayout يتيح لك StackLayout التحكم في المسافة بين كل عنصر فرعي باستخدام خاصية Spacing القيمة الافتراضية هي صفر وحدات The default value is zero units ولكن يمكنك تعيينها على أي قيمة مناسبة لك، فيما يلي مثال لتعيين خاصية Spacing إلى 30 في XAML

```
<StackLayout Spacing="30">
    <BoxView Color="Silver" HeightRequest="40" />
    <BoxView Color="Blue" HeightRequest="40" />
    <BoxView Color="Gray" HeightRequest="40" />
</StackLayout>
```

توضح لقطة الشاشة التالية كيفية عرض واجهة المستخدم على Android



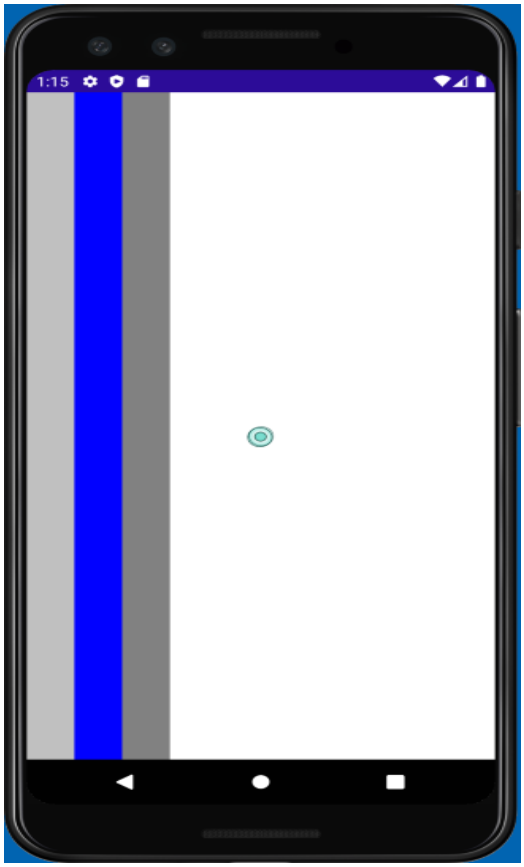
كيفية تعيين اتجاه set the orientation of a StackLayout

يتيح لك StackLayout ترتيب العناصر الفرعية إما في عمود أو صف. يمكنك التحكم في هذا السلوك من خلال تعيين خاصية Orientation حتى الآن، كنا نعرض vertical StackLayout رأسيًا فقط.

الوضع الرأسي أو العمودي Vertical هو الافتراضي، أما إذا كنت تريد تعيين الاتجاه Orientation بشكل صريح على الوضع الرأسي Vertical فهذا الأمر متروك لك. يفضل بعض المبرمجين إعدادًا صريحًا لجعل التعليمات أكثر توثيقًا ذاتيًا. فيما يلي مثال على تعيين الخاصية Orientation إلى أفقي Horizontal في XAML

```
<StackLayout x:Name="stack" Orientation="Horizontal">
    <BoxView Color="Silver" WidthRequest="40"/>
    <BoxView Color="Blue" WidthRequest="40"/>
    <BoxView Color="Gray" WidthRequest="40"/>
</StackLayout>
```

ملاحظة: كما هو موضح في التمرين السابق، يؤدي التغيير في اتجاه StackPanel إلى تجاهل خصائص HeightRequest لكل BoxView بدلاً من ذلك، يمكنك تعيين WidthRequest توضح لقطة الشاشة التالية كيفية عرض واجهة المستخدم على جهاز Android



تعيين تخطيطات طريقة عرض view's LayoutOptions في StackLayout

تحتوي كل view على خاصيتي VerticalOptions and HorizontalOptions يمكنك استخدام هذه الخصائص لتعيين موضع طريقة العرض داخل منطقة العرض المستطيلة التي توفرها لوحة التخطيط.

كما ذكرنا سابقًا، مع StackLayout يعتمد سلوك خيارات التخطيط LayoutOptions على خاصية Orientation في StackLayout يستخدم StackLayout خاصية LayoutOptions في الاتجاه المعاكس لـ Orientation بشكل افتراضي، لا يتم تخصيص أي مساحة إضافية لعنصر في تخطيط المكدس في نفس اتجاه Orientation تخطيط المكدس، لا يؤدي تعيين موضع لهذا الاتجاه في هذه الحالة الافتراضية إلى تغيير عرض العنصر. ومع ذلك، هناك تغيير في العرض عندما يتم الجمع بين الموضع والتوسع position is combined with expansion

ماذا عن التوسع expansion

تذكر من درس سابق أن بنية LayoutOptions تحتوي على خاصية bool تسمى Expands والتي أصبحت قديمة الآن في MAUI .NET عند تعيين VerticalOptions and HorizontalOptions تلاحظ LayoutOptions الخاصة بـ StartAndExpand, CenterAndExpand, EndAndExpand, and FillAndExpand إذا قمت بتعيين LayoutOptions على أحد هذه الخيارات AndExpand

فسيتم تجاهل الخيار ويتم استخدام الجزء الأول من LayoutOptions مثل Start أو Center أو End أو Fill إذا كنت تقوم بالترحيل من Xamarin.Forms فيجب إزالة جميع خصائص AndExpand من هذه الخصائص. لاحقًا في الوحدة، سنرى كيفية تحقيق وظائف مماثلة عندما نتعلم عن Grid

التخطيطات المُحسَّنة Optimized StackLayouts

كما ذكرنا سابقًا، فإن VerticalStackLayout and HorizontalStackLayout عبارة عن عناصر تحكم StackLayout مُحسَّنة ذات اتجاهات محددة مسبقًا، نوصيك باستخدام عناصر التحكم هذه كلما يكون ذلك ممكنًا، للحصول على أفضل أداء للتخطيط.

تتمتع هذه التخطيطات بوظائف LayoutOptions and Spacing التي تحتوي عليها التخطيطات العادية StackLayout

```
<VerticalStackLayout Spacing="30">
    <BoxView Color="Silver" HeightRequest="40" />
```

```
    <BoxView Color="Blue" HeightRequest="40" />
    <BoxView Color="Gray" HeightRequest="40"/>
</VerticalStackLayout>

<HorizontalStackLayout Spacing="30">
    <BoxView Color="Silver" WidthRequest="40" />
    <BoxView Color="Blue" WidthRequest="40" />
    <BoxView Color="Gray" WidthRequest="40" />
</HorizontalStackLayout>
```

سؤال بسيط:

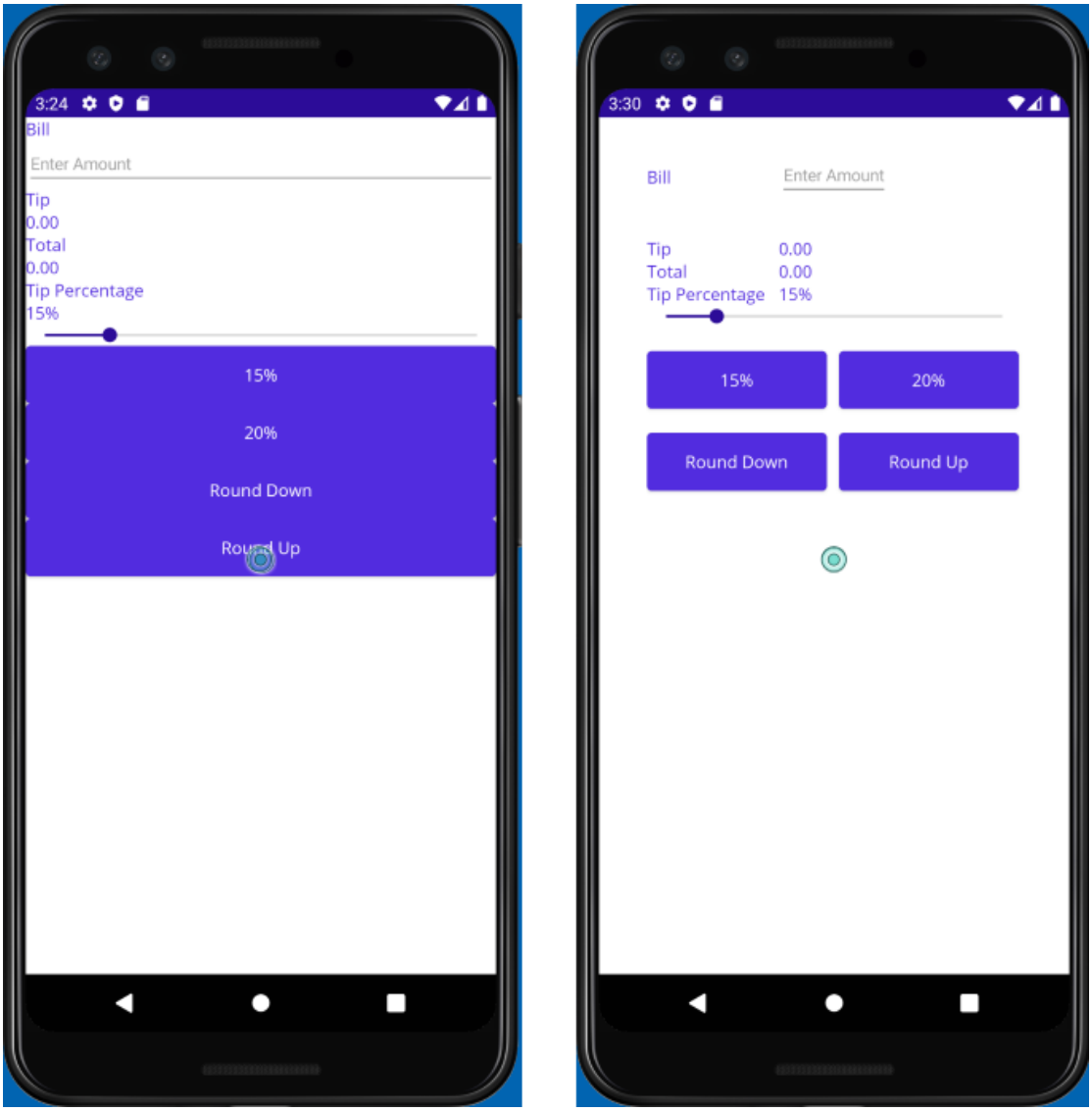
- كيف يمكنك إضافة طرق عرض تابعة child views إلى StackLayout

حل الاختبار:

- يمكنك إضافة طرق عرض تابعة child views إلى StackLayout باستخدام XAML أو في صفحة التعليمات البرمجية الخلفية باستخدام C#

٥ تمرين - استخدام StackLayout لإنشاء واجهة مستخدم user interface

في هذا التمرين، يمكنك استخدام حاويات StackLayout المتداخلة، لترتيب طرق العرض views في واجهة المستخدم (UI) تظهر لقطة الشاشة الأولى التخطيط الذي تم تنفيذه بواسطة المشروع المبدئي starter project والثانية تعرض تخطيط المشروع المكتمل. مهمتك هي استخدام StackLayout والحاويات LayoutOptions وتحويل المشروع starter إلى النسخة الكاملة.



استكشاف الحل the starter solution

يحتوي starter solution على تطبيق حاسبة للإكراميات "بقشيش" يعمل بكامل طاقته، ابدأ باستكشاف واجهة المستخدم لفهم ما يفعله التطبيق.

١- باستخدام Visual Studio افتح starter solution من المجلد exercise2/TipCalculator الذي نسخته في بداية التمرين السابق.

٢- إنشاء التطبيق وتشغيله Build and run على نظام التشغيل المفضل لديك.

٣- أدخل رقمًا في مربع النص واستخدم التطبيق لمعرفة كيفية عمله.

٤- جرب أزرار مبلغ التلميح buttons وشريط التمرير slider

٥- عند الانتهاء، أغلق التطبيق.

٦- افتح MainPage.xaml لاحظ أنه يتم وضع جميع طرق العرض في VerticalStackLayout واحد، كما هو موضح في علامات XAML markup التالية:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
      xmlns:local="clr-namespace:TipCalculator"
      x:Class="TipCalculator.MainPage">
    <VerticalStackLayout>

      <Label Text="Bill" />
      <Entry x:Name="billInput" Placeholder="Enter
Amount" Keyboard="Numeric" />

      <Label Text="Tip"   />
      <Label x:Name="tipOutput" Text="0.00" />

      <Label Text="Total" />
      <Label x:Name="totalOutput" Text="0.00" />

      <Label Text="Tip Percentage" />
      <Label x:Name="tipPercent" Text="15%" />
      <Slider x:Name="tipPercentSlider" Minimum="0"
Maximum="100" Value="15" />

      <Button Text="15%" Clicked="OnNormalTip" />
      <Button Text="20%" Clicked="OnGenerousTip" />

      <Button x:Name="roundDown" Text="Round Down" />
      <Button x:Name="roundUp"   Text="Round Up" />

    </VerticalStackLayout>
</ContentPage>
```

إصلاح واجهة المستخدم Fix the UI

الآن بعد أن رأيت تشغيل التطبيق، اجعله يبدو أفضل عن طريق إضافة حاويات تخطيط HorizontalStackLayout الهدف هو جعل التطبيق يبدو مثل لقطة الشاشة في بداية التمرين.

١- افتح الملف MainPage.xaml

٢- أضف 40 وحدة للحشو units of padding و10 وحدات للمسافة units of spacing إلى VerticalStackLayout

```
<VerticalStackLayout Padding="40" Spacing="10">
```

٣- أضف HorizontalStackLayout لتجميع الملصق Label الذي يحمل اسم Bill مع حقل الإدخال Entry أسفله، اضبط خاصية Spacing على 10

٤- اضبط WidthRequest لملصق Bill Label على 100 وخاصية VerticalOptions على Center تضمن هذه التغييرات محاذاة الملصق Label رأسياً مع حقل Entry

```
<HorizontalStackLayout Spacing="10">
   <Label Text="Bill" WidthRequest="100"
    VerticalOptions="Center"/>
   <Entry ... />
</HorizontalStackLayout>
```

٥- أضف HorizontalStackLayout آخر لتجميع Label الذي يحمل اسم Tip مع Label المسمى tipOutput اضبط خاصية Spacing على 10 وخاصية Margin على `0,20,0,0`

٦- اضبط خاصية WidthRequest ل Tip Label على 100

```
<HorizontalStackLayout Margin="0,20,0,0"
Spacing="10">
    <Label Text="Tip" WidthRequest="100" />
    <Label .../>
</HorizontalStackLayout>
```

٧- استخدم HorizontalStackLayout لتجميع Label المسمى Total مع Label المسمى totalOutput اضبط خاصية Spacing على 10

٨- تعيين WidthRequest ل Total Label إلى 100

```
<HorizontalStackLayout Spacing="10">
```

```
    <Label Text="Total" WidthRequest="100" />
    <Label .../>
</HorizontalStackLayout>
```

٩- أضف HorizontalStackLayout آخر لتجميع Label المسمى Tip Percentage مع Label المسمى tipPercent

١٠- اضبط خاصية VerticalOptions لـ HorizontalStackLayout على End واضبط خاصية Spacing على 10

١١- اضبط WidthRequest لـ Tip Percentage Label على 100

```
<HorizontalStackLayout VerticalOptions="End"
Spacing="10">
 <Label Text="Tip Percentage" WidthRequest="100"/>
 <Label ... />
</HorizontalStackLayout>
```

١٢- استخدم HorizontalStackLayout لتجميع Button الذي يحمل التسمية التوضيحية **15%** وButton الذي يحمل التسمية التوضيحية **20%**

١٣- اضبط خاصية `Margin` في StackLayout على `0,20,0,0` وخاصية `Spacing` على 10

```
<HorizontalStackLayout  Margin="0,20,0,0"
Spacing="10">
    <Button Text="15%" ... />
    <Button Text="20%" ... />
</HorizontalStackLayout>
```

١٤- أضف HorizontalStackLayout نهائي، لتجميع Button الذي يحمل التسمية التوضيحية **Round Down** وButton الذي يحمل التسمية التوضيحية **Round Up..** اضبط خاصية Margin في StackLayout على `0,20,0,0` وخاصية Spacing على 10

```
<HorizontalStackLayout Margin="0,20,0,0" Spacing="10">
        <Button ... Text="Round Down" />
        <Button ... Text="Round Up" />
</HorizontalStackLayout>
```

١٥- في جميع الأزرار الأربعة buttons اضبط خاصية HorizontalOptions على Center وخاصية WidthRequest على 150 على سبيل المثال:

```
<Button Text="15%" WidthRequest="150"
HorizontalOptions="Center" ... />
```

يجب أن تبدو علامات لغة (XAML) الكاملة لصفحة content page كما يلي:

```
<?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"
     xmlns:local="clr-namespace:TipCalculator"
     x:Class="TipCalculator.MainPage">

    <VerticalStackLayout Padding="40" Spacing="10">

        <HorizontalStackLayout Spacing="10">
            <Label Text="Bill" WidthRequest="100"
VerticalOptions="Center" />
            <Entry x:Name="billInput"
Placeholder="Enter Amount" Keyboard="Numeric" />
        </HorizontalStackLayout>

        <HorizontalStackLayout Margin="0,20,0,0"
Spacing="10">
            <Label Text="Tip"  WidthRequest="100" />
            <Label x:Name="tipOutput" Text="0.00" />
        </HorizontalStackLayout>

        <HorizontalStackLayout Spacing="10">
           <Label Text="Total" WidthRequest="100"/>
           <Label x:Name="totalOutput" Text="0.00" />
        </HorizontalStackLayout>

        <HorizontalStackLayout VerticalOptions="End"
Spacing="10">
            <Label Text="Tip Percentage"
WidthRequest="100"/>
            <Label x:Name="tipPercent" Text="15%" />
        </HorizontalStackLayout>

        <Slider x:Name="tipPercentSlider" Minimum="0"
Maximum="100" Value="15" />

        <HorizontalStackLayout Margin="0,20,0,0"
Spacing="10">
```

```
            <Button Text="15%" Clicked="OnNormalTip"
WidthRequest="150" HorizontalOptions="Center"/>
            <Button Text="20%"
Clicked="OnGenerousTip"  WidthRequest="150"
HorizontalOptions="Center"/>
        </HorizontalStackLayout>

        <HorizontalStackLayout Margin="0,20,0,0"
Spacing="10">
            <Button x:Name="roundDown" Text="Round
Down" WidthRequest="150" HorizontalOptions="Center"/>
            <Button x:Name="roundUp"   Text="Round
Up" WidthRequest="150" HorizontalOptions="Center"/>
        </HorizontalStackLayout>

    </VerticalStackLayout>

</ContentPage>
```

فحص النتائج

شغل التطبيق مرة أخرى، وانظر في الاختلافات في واجهة المستخدم، تحقق من محاذاة عناصر التحكم بشكل صحيح، ومن أنها ذات حجم ومسافة مناسبة.

لقد استخدمت حاويات VerticalStackLayout وHorizontalStackLayout لتحسين جماليات واجهة مستخدم موجودة، هذه التخطيطات هي أبسط لوحات التخطيط، ولكنها قوية بما يكفي لإنتاج واجهة مستخدم معقولة.

٦ ترتيب طرق العرض views باستخدام الشبكة Grid

لنفترض أنك تقوم بإنشاء صفحة تعرض الصور في شبكة `7x5 grid` من الممكن إنشاء هذه الصفحة مع حاويات متعددة أفقية وعمودية horizontal and vertical StackLayout ولكن سيكون مملاً للتعليمة البرمجية ويمكن أن يسبب مشاكل في الأداء، بسبب متطلبات الذاكرة، والمعالجة من لوحات تخطيط متعددة، تعتبر لوحة التخطيط Grid خياراً أفضل لواجهات المستخدم التي تحتاج صفوف وأعمدة.

في هذا الدرس، ستتعلم كيفية تعريف Grid وتحديد موضع طرق العرض position views داخل خلاياها.

ما هي Grid

Grid هي لوحة تخطيط تتكون من صفوف وأعمدة، يبين الرسم التوضيحي التالي عرض تصوري Grid

يمكنك وضع views في الخلايا cells التي تم إنشاؤها من تقاطع الصفوف والأعمدة rows and columns على سبيل المثال، إذا قمت بإنشاء Grid تحتوي على ثلاثة أعمدة وصفّان، فهناك ست خلايا متوفرة لـ views يمكن أن تكون الصفوف والأعمدة أحجامًا مختلفة، أو يمكن تعيينها للتكيف تلقائيًا مع حجم العناصر التابعة الموجودة داخلها. يمكن أن تشغل طرق العرض التابعة Child views خلية واحدة أو تمتد عبر العديد من الخلايا. هذه المرونة تجعل Grid خيارًا جيدًا للوحة تخطيط الجذر root layout panel للعديد من التطبيقات.

كيفية تحديد الصفوف والأعمدة لخطوط شبكة rows and columns of a Grid

عند إنشاء Grid يمكنك تعريف كل صف وعمود على حدة، يمنحك هذا النظام التحكم الكامل في ارتفاع كل صف وعرض كل عمود، كل Grid لديه مجموعة من العناصر objects التي تحدد شكل `RowDefinition` and `ColumnDefinition` خطوط الشبكة، يمكنك ملء هذه المجموعات بمثيلات `RowDefinition` and `ColumnDefinition` كل منها يمثل صفاً أو عمودًا في واجهة المستخدم الخاصة بك.

فيما يلي مقتطفات من التعليمات البرمجية التي تظهر تعريفات الفئة class لـ `RowDefinition` and `ColumnDefinition`

```
public sealed class RowDefinition : ...
{
    ...
    public GridLength Height { get; set; }
}
```

.....

```
public sealed class ColumnDefinition : ...
{
    ...
    public GridLength Width { get; set; }
}
```

لاحظ أن RowDefinition له خاصية تسمى `Height` وأن ColumnDefinition له خاصية تسمى `Width` يمكنك استخدام هذه الخصائص لتعيين ارتفاع صف وعرض عمود، كما هو موضح في المقاطع التالية.

ما هو GridLength

نوع البيانات لخصائص `Width and Height` هو GridLength يحتوي هذا النوع على خاصيتين `GridUnitType and Value` فيما يلي مقتطف من التعليمات البرمجية يوضح جزءًا من تعريف النوع the type definition

```
public struct GridLength
{
    public GridUnitType GridUnitType { get; }
    public double Value { get; }
}
```

يمكنك تعيين الخاصية GridUnitType إلى إحدى القيم التالية:

- Absolute
- Auto
- Star

دعونا نلق نظرة فاحصة على كل من هذه القيم.

نوع الشبكة المطلقة Absolute GridUnitType

يحدد Absolute أن يكون حجم الصف أو العمود ثابتًا، يمكنك استخدام الخاصية Value للإشارة إلى الحجم size فيما يلي مثال يوضح كيفية تعيين ارتفاع الصف ليكون بحجم ثابت يبلغ 100 من وحدة الجهاز في C# لاحظ كيفية استخدام الدالة GridLength الإنشائية التي تأخذ قيمة رقمية numeric value يقوم هذا المنشئ بتعيين GridUnitType إلى Absolute نيابةً عنك تلقائيًا.

```
var row = new RowDefinition() { Height = new GridLength(100) };
```

في (XAML) ما عليك سوى توفير قيمة رقمية numeric value يستدعي محلل XAML محول نوع type converter لإنشاء المثيل GridLength فيما يلي مثال يوضح نفس الشيء في XAML

```
<RowDefinition Height="100" />
```

Auto GridUnitType

يقوم `Auto` تلقائيًا بتغيير حجم الصف أو العمود ليناسب الطرق الفرعية child views يقوم Grid بمسح كافة طرق العرض التابعة في هذا الصف أو العمود، ويحدد أكبر عرض the largest view ثم يجعل الصف أو العمود كبيرًا بما يكفي ليناسب هذا العرض الفرعي child view

عند إنشاء تعريف صف row definition في التعليمات البرمجية، يتم تجاهل القيمة الرقمية numeric value يمكنك استخدام أي قيمة. فيما يلي مثال يوضح كيفية تعيين ارتفاع الصف ليكون بحجم تلقائي في C# لاحظ أننا اخترنا القيمة 1 بشكل تعسفي.

```
var row = new RowDefinition() { Height = new GridLength(1, GridUnitType.Auto) };
```

في XAML يمكنك استخدام القيمة Auto هنا مثال يظهر نفس الشيء في XAML

```
<RowDefinition Height="Auto" />
```

Star GridUnitType

تمنحك Star إمكانية تحديد الحجم بشكل متناسب، في التحجيم النسبي proportional sizing يحدد إجمالي المساحة المتوفرة والنسبة التي يطلبها كل صف أو عمود لتحديد الحجم، غالبًا ما يطلق عليه التحجيم النجمي star sizing بدلاً من تغيير الحجم النسبي proportional sizing

دعونا نلقي نظرة على عملية استخدام التحجيم النسبي proportional sizing للصفوف في grid

١- Determine the available space تحديد المساحة المتوفرة: تقوم Grid بمسح كافة الصفوف التي لا تستخدم تغيير الحجم بالنجوم star sizing وتقوم بجمع ارتفاع كل هذه الصفوف وطرح هذا الإجمالي من ارتفاع الشبكة نفسها، توفر هذه العملية الحسابية مقدار المساحة المتوفرة لكافة الصفوف ذات حجم النجمة star-sized rows

٢- Divide the available space تقسيم المساحة المتوفرة: ثم تقسم Grid المساحة المتوفرة بين كافة الصفوف ذات حجم النجمة استنادًا إلى إعداد القيمة Value لكل صف row فكر في الخاصية Value كمضاعف يحدد النسبة بين كافة الصفوف المعرفة بالحجم النجمي star sized على سبيل المثال، إذا كان لدينا صفان بحجم نجمة، كلاهما 1 كمضاعف، فسيتم تقسيم المساحة المتوفرة بالتساوي بينهما، ولكن إذا كان أحدهما يحتوي على 2 كقيمة، فسيحصل على مساحة أكبر بمرتين من الآخر.

فيما يلي مثال يوضح كيف يمكنك تعيين ارتفاع الصف height of a row ليكون `2` `Star` في C#:

```
var row = new RowDefinition() { Height = new GridLength(2, GridUnitType.Star) };
```

في XAML يمكنك استخدام الرمز `*` symbol لتمثيل تغيير الحجم بالنجوم star sizing يمكنك ضم `*` and `value` في سلسلة واحدة single string وسيقوم محول النوع type converter بإنشاء `GridLength` لك، إليك نفس المثال في XAML

```
<RowDefinition Height="2*" />
```

مجموعات الشبكة Grid collections

بعد تعريف الصفوف والأعمدة rows and columns باستخدام RowDefinition and ColumnDefinition يمكنك إضافتها إلى Grid ثم يمكنك استخدام خصائص مجموعة RowDefinitions and ColumnDefinitions collection properties في Grid يتم عادةً ملء هذه المجموعات باستخدام XAML

يوضح هذا المثال كيفية تعريف أربعة صفوف وإضافتها إلى Grid باستخدام الخاصية RowDefinitions

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="100" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="1*" />
        <RowDefinition Height="2*" />
    </Grid.RowDefinitions>
    ...
</Grid>
```

يمكن اختصار هذا التعريف إلى:

```
<Grid RowDefinitions="100, Auto, 1*, 2*">
    ...
</Grid>
```

إن لغة XAML المستخدمة في تعريف الأعمدة تشبه لغة XAML السابقة، باستثناء أنك ستستخدم `ColumnDefinitions` وتحدد العرض `Width`

في وقت التشغيل XAML تنتج Grid بأربعة صفوف rows يحتوي الصف الأول على ارتفاع ثابت `height of 100 device units` الصف الثاني له ارتفاع أطول view في الصف row يستخدم الصفان الثالث والرابع تحجيم النجوم star sizing مما يعني أنهما يأخذان المساحة المتاحة المتبقية، ويقسمانها بشكل متناسب بناءً على مضاعف القيمة Value multiplier نظرا لأن الصف الثالث هو `1*` والصف الرابع هو `2*` فإن الصف الرابع يبلغ ارتفاعه ضعف ارتفاع الصف الثالث.

الحجم الافتراضي للصفوف والأعمدة Row and column default size

الحجم الافتراضي للصفوف والأعمدة هو الحجم `1*` على سبيل المثال، انظر إلى XAML التالي.

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    ...
</Grid>
```

يمكن اختصار هذا التعريف إلى:

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

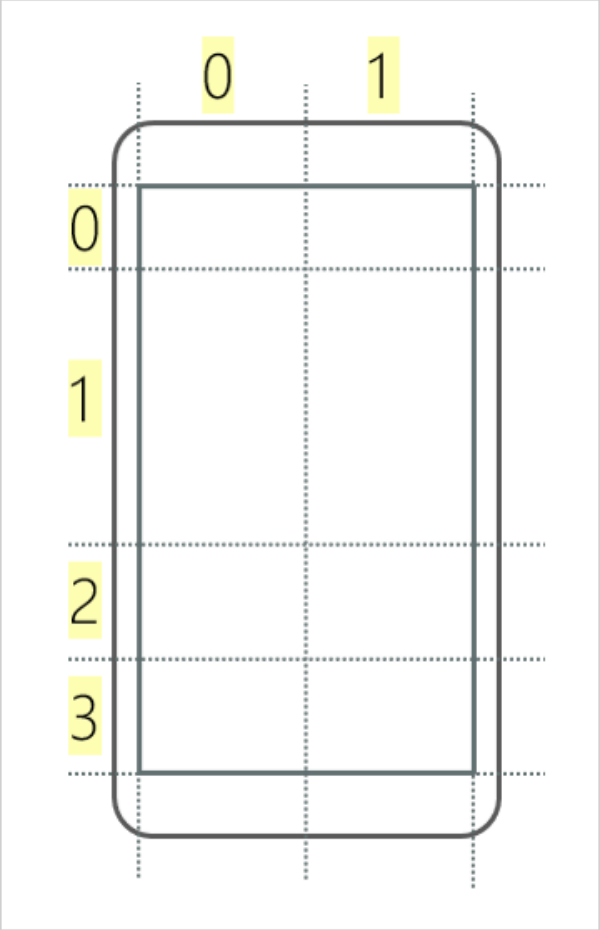
نظرا لعدم تحديد أحجام لأي من الصفوف أو الأعمدة، يتم تطبيق `1*` عليها جميعا، في وقت التشغيل، ينشئ هذا التكوين Grid موحدة، ما يعني أن جميع الصفوف لها نفس الارتفاع height وجميع الأعمدة لها نفس العرض width

كيفية إضافة طرق عرض views إلى الشبكة Grid

عند إضافة view إلى Grid فإنك تضيفه إلى خلية محددة specific cell يتم إنشاء الخلايا في المواضع حيث تتقاطع الصفوف والأعمدة، لوضع view في خلية، تحتاج إلى معرفة موقع الخلية location of the cell يمكنك استخدام مزيج من رقم الصف row number ورقم العمود column number لتحديد الخلية cell

ترقيم الصفوف والأعمدة Row and column numbering

يبدأ ترقيم الصفوف والأعمدة من صفر، البداية من الزاوية العلوية اليسرى، هنا رسم توضيحي يظهر الترقيم لـ Grid بأربعة صفوف وعمودين:



على سبيل المثال، إذا أردنا إضافة view إلى الخلية في أسفل اليمين bottom-right cell فسنقول إن موضع view هو `row 3 column 1`

إضافة view إلى Grid باستخدام الخصائص المرفقة attached properties

تحتاج إلى نهج لتحديد رقم الصف والعمود ل view عندما نضيفه إلى Grid أحد الحلول هو تعريف خصائص الصف والعمود Row and Column في `View base class` بحيث تتمكن من تحديد الموضع في view مباشرةً، هذه التقنية ستعمل، لكنها ليست النهج الأكثر كفاءة، لن تكون Views دائمًا في Grid لذا في بعض الأحيان لن تكون هناك حاجة إلى هذه الخصائص، النهج الأفضل هو استخدام الخصائص المرفقة attached properties

الخاصية المرفقة attached property هي خاصية معرفة في one class ولكن يتم تعيينها على كائنات من أنواع أخرى objects of other types

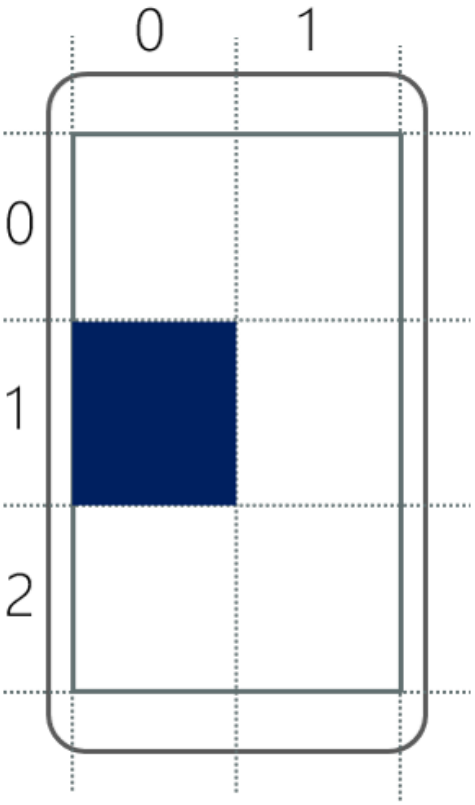
فكر في الخصائص المرفقة على أنها مجموعة من أزواج المفتاح والقيمة key-value التي تشكل جزءًا من view عند إضافة view إلى Grid يمكنك تحديد الصف والعمود. باستخدام الخصائص المرفقة attached properties يمكنك إضافة زوج المفتاح والقيمة key-value مع key Grid.Row والقيمة التي تحدد رقم الصف، عندما

تكون Grid جاهزة لموضع position the view فإنها تتحقق من المجموعة لمعرفة ما إذا كان هناك مفتاح يسمى Grid.Row إذا كان هناك، تستخدم Grid القيمة لوضع position the view

يوضح هذا المثال كيفية إنشاء Grid وإضافة طريقة عرض view باستخدام الخصائص المرفقة attached properties

```
<Grid RowDefinitions="*, *, *"
ColumnDefinitions="*, *">

    <BoxView Grid.Row="1" Grid.Column="0"
Color="Navy" />

</Grid>
```

في هذا المثال Grid.Row=1 and Grid.Column=0 عبارة عن أزواج مفتاح-قيمة key-value pairs يتم إضافتها إلى مجموعة داخلية من BoxView تستخدم Grid هذه القيم لتحديد المكان الذي يجب وضع View فيه. فيما يلي الشكل الذي ستبدو عليه Grid إذا قمت بتشغيل التطبيق على جهاز.

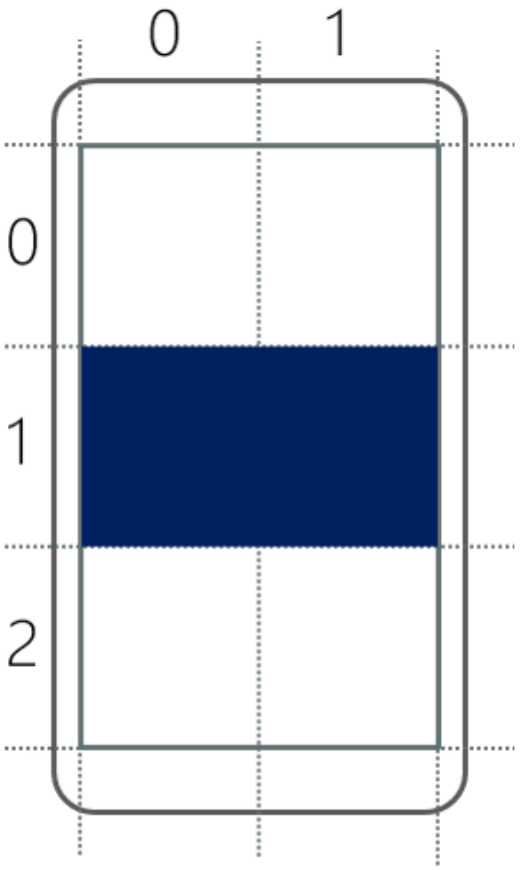


كيفية جعل طريقة عرض view تمتد على عدة صفوف أو أعمدة span multiple rows or columns

هناك خاصيتان مرفقتان يجب أن تكون على علم بهما `Grid.RowSpan` and `Grid.ColumnSpan` تحدد هذه الخصائص عدد الصفوف أو الأعمدة التي يجب أن تشغلها طريقة العرض view على سبيل المثال، انظر إلى XAML التالي:

```
<Grid RowDefinitions="*, *, *"
ColumnDefinitions="*, *">

    <BoxView Grid.Row="1" Grid.Column="0"
Grid.ColumnSpan="2" Color="Navy" />

</Grid>
```

لاحظ أن هذا المثال يعين `ColumnSpan` إلى 2 تشغل view هذه عمودين بدءاً من `Column 0` إليك ما سيبدو عليه Grid إذا قمت بتشغيل التطبيق على جهاز.



اختبار بسيط:

١- ما هو الغرض من Star GridUnitType

٢- ما هو الغرض من الخاصية RowSpan

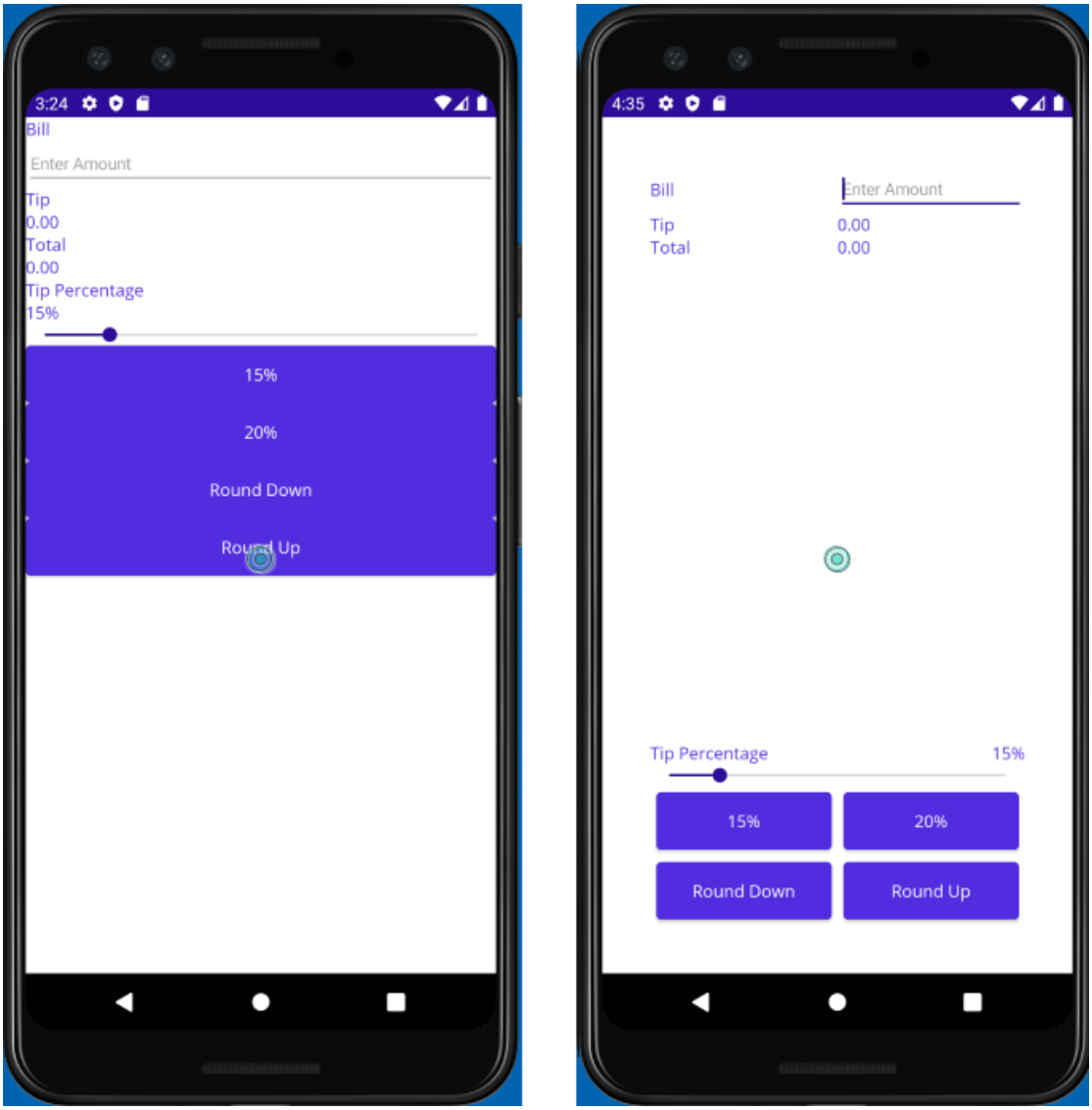
حل الاختبار:

١- التحجيم النسبي Proportional sizing لعمود أو صف، يتم تحديد الحجم عن طريق حساب إجمالي المساحة المتوفرة، والنسبة التي يطلبها كل صف أو عمود. ويشار إليه **star or ***

٢- الخاصية RowSpan تحدد قيمتها عدد الصفوف التي سوف تشغلها طريقة العرض view

٧ تمرين - استخدام Grid لإنشاء واجهة مستخدم user interface

في هذا التمرين، يمكنك استخدام Grid لترتيب طرق العرض في واجهة المستخدم (UI) تبدأ بإصدار آخر من مشروع TipCalculator وتضبطه لجعل واجهة المستخدم أكثر سهولة، يمكنك أيضا نقل الأزرار إلى أسفل الصفحة، هذه المرة يمكنك استخدام تخطيط Grid بدلاً من استخدام VerticalStackLayout and HorizontalStackLayout تظهر الصورة التالية واجهة المستخدم الأولية، وواجهة المستخدم النهائية التي تنتج عن اتباع الخطوات الواردة في هذا التمرين:



افتح the starter solution

يحتوي الحل المبدئي على تطبيق حاسبة الإكراميات TipCalculator مكتمل الوظائف.

١- استخدم Visual Studio لفتح the starter solution في المجلد exercise3/TipCalculator الذي نسخته في بداية التمرين السابق.

٢- افتح MainPage.xaml لاحظ أنه يتم عرض جميع views باستخدام لوحة VerticalStackLayout واحدة.

```
<?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"
        xmlns:local="clr-namespace:TipCalculator"
        x:Class="TipCalculator.MainPage">

 <VerticalStackLayout>

    <Label Text="Bill" />
    <Entry x:Name="billInput" Placeholder="Enter
Amount" Keyboard="Numeric" />

    <Label Text="Tip"   />
    <Label x:Name="tipOutput" Text="0.00" />

    <Label Text="Total" />
    <Label x:Name="totalOutput" Text="0.00" />

    <Label Text="Tip Percentage" />
    <Label x:Name="tipPercent" Text="15%" />
    <Slider x:Name="tipPercentSlider" Minimum="0"
Maximum="100" Value="15" />

    <Button Text="15%" Clicked="OnNormalTip" />
    <Button Text="20%" Clicked="OnGenerousTip" />

    <Button x:Name="roundDown" Text="Round Down" />
    <Button x:Name="roundUp"   Text="Round Up" />

 </VerticalStackLayout>

</ContentPage>
```

إنشاء تخطيط شبكة Grid layout

١- استبدل لوحة التخطيط VerticalStackLayout إلى Grid مع ترك هامش أو حشو padding of 40 units

٢- حدد سبعة صفوف وعمودين لـ Grid أجعل كافة الصفوف ذات حجم تلقائي `Auto size` باستثناء الصف الرابع، يجب أن يستخدم الصف الرابع `Star` حتى يحصل على كل المساحة المتبقية المتاحة في Grid استخدم Star sizing لكلا العمودين.

```
<Grid RowDefinitions="Auto, Auto, Auto, *,
Auto, Auto, Auto"
      ColumnDefinitions="*, *"
      Padding="40">
</Grid>
```

وضع طرق العرض في الخلايا Position the views in the cells

أضف إعدادات لـ Grid.Row and Grid.Column لكل views وتعيينها إلى الخلية المناسبة في Grid استخدم لقطة الشاشة التالية لمساعدتك في تحديد مكان موضع كل view

يوضح المثال التالي كيفية تعيين الموضع **Bill** Label وطريقة العرض billInput Entry

```
...
<Label Text="Bill" Grid.Row="0" Grid.Column="0"/>
<Entry x:Name="billInput" Placeholder="Enter
Amount" Keyboard="Numeric" Grid.Row="0"
Grid.Column="1"/>
...
```

٢- قم بمحاذاة Bill Label and Entry عن طريق تعيين الخاصية VerticalOptions إلى Center على Label

٣- إضافة إعدادات Grid.ColumnSpan لـ Slider بحيث يمتد عمودين:

```
<Slider ... Grid.ColumnSpan="2" ... />
```

٤- حدد Label المسمى Tip Percentage قم بتعيينه بحيث يشغل الموضع الأيسر السفلي lower-left position في مستطيله:

```
<Label Text="Tip Percentage" VerticalOptions="End"
HorizontalOptions="Start" ... />
```

٥- حدد موقع Label المسمى tipPercent قم بتعيينه بحيث يحتل الموضع السفلي الأيمن lower-right في مستطيله:

```
<Label  x:Name="tipPercent"  VerticalOptions="End"
HorizontalOptions="End" ... />
```

٦- قم بتعيين الخاصية Margin لجميع buttons الأربعة إلى 5

يجب أن تبدو علامات لغة XAML الكاملة للصفحة كما يلي:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:local="clr-namespace:TipCalculator"
        x:Class="TipCalculator.MainPage">
    <Grid RowDefinitions="Auto, Auto, Auto, *, Auto,
Auto, Auto"
          ColumnDefinitions="*, *"
          Padding="40">
```

```
        <Label Text="Bill" VerticalOptions="Center"
Grid.Row="0" Grid.Column="0"/>
        <Entry x:Name="billInput" Placeholder="Enter
Amount" Keyboard="Numeric" Grid.Row="0"
Grid.Column="1"/>

        <Label Text="Tip" Grid.Row="1"
Grid.Column="0"/>
        <Label x:Name="tipOutput" Text="0.00"
Grid.Row="1" Grid.Column="1"/>

        <Label Text="Total" Grid.Row="2"
Grid.Column="0"/>
        <Label x:Name="totalOutput" Text="0.00"
Grid.Row="2" Grid.Column="1"/>

        <Label Text="Tip Percentage"
VerticalOptions="End" HorizontalOptions="Start"
Grid.Row="3" Grid.Column="0"/>
        <Label x:Name="tipPercent" Text="15%"
VerticalOptions="End" HorizontalOptions="End"
Grid.Row="3" Grid.Column="1"/>
        <Slider x:Name="tipPercentSlider" Minimum="0"
Maximum="100" Value="15" Grid.Row="4" Grid.Column="0"
Grid.ColumnSpan="2"/>

        <Button Text="15%" Clicked="OnNormalTip"
Margin="5" Grid.Row="5" Grid.Column="0"/>
        <Button Text="20%" Clicked="OnGenerousTip"
Margin="5" Grid.Row="5" Grid.Column="1"/>

        <Button x:Name="roundDown" Margin="5"
Text="Round Down" Grid.Row="6" Grid.Column="0"/>
        <Button x:Name="roundUp"   Margin="5"
Text="Round Up" Grid.Row="6" Grid.Column="1"/>

    </Grid>
</ContentPage>
```

فحص النتائج

قم بتشغيل التطبيق وانظر إلى الاختلافات في واجهة المستخدم UI لقد استخدمت Grid لتحسين جماليات واجهة المستخدم الموجودة، Grid أقوى من StackLayout على وجه الخصوص Grid تسهل محاذاة طرق العرض views عبر الصفوف rows

٨ الملخص

من الصعب تصميم واجهة مستخدم تبدو مشابهة لكل الأجهزة. يمكن أن يكون لكل جهاز عامل شكل مختلف وحجم size وكثافة بكسل pixel density ستحتاج إلى كتابة مزيد من التعليمات البرمجية للتأكد من أن تطبيقك يبدو رائعًا على جميع الأجهزة.

توفر واجهة مستخدم التطبيق متعددة المنصات .NET Multi-platform App UI (MAUI) لوحات تخطيط للمساعدة في حل هذه المشكلة. باستخدام StackLayout, VerticalStackLayout, HorizontalStackLayout, Grid لديك سيطرة واسعة على ترتيب طرق العرض views

باستخدام استراتيجيات التخطيط التي تعلمتها هنا، يمكنك تطوير تطبيقات تتكيف مع جميع أجهزة المستخدم المتعددة.

معرفة المزيد عناصر التحكم controls

الوحدة الرابعة

تصميم صفحات .NET MAUI XAML. متناسقة باستخدام الموارد المشتركة shared resources

التعرف على كيفية استخدام الموارد المشتركة shared resources والأنماط styles في .NET MAUI XAML.

الأهداف التعليمية

في نهاية هذه الوحدة، تتمكن من:

- إنشاء موارد ثابتة static resources واستخدامها في واجهة مستخدم MAUI XAML
- إنشاء الموارد التفاعلية أو الحيوية dynamic resources واستخدامها
- إنشاء واجهة مستخدم متناسقة باستخدام الأنماط styles
- إنشاء موارد على مستوى التطبيق application-wide resources واستخدامها
- تطبيق خيارات إمكانية وصول ذوي الاحتياجات الخاصة للمستخدم باستخدام الأنماط المضمنة built-in styles

محتويات الوحدة:

١- المقدمة

٢- تحديد الموارد واستخدامها Define and use resources

٣- تمرين: استخدام الموارد على مستوى الصفحة page-level resources

٤- استخدام الموارد التفاعلية وتحديثها update dynamic resources

٥- تمرين: استخدام الموارد التفاعلية dynamic resources لتحديث العناصر update elements

٦- إنشاء واجهة مستخدم متسقة consistent UI باستخدام الأنماط styles

٧- تمرين: إنشاء نمط وتطبيقه Create and apply a style

٨- إنشاء موارد على مستوى التطبيق application-wide resources واستخدامها

٩- تمرين: استخدام الموارد على مستوى التطبيق application-wide resources

١٠- الملخص

١ المقدمة

يؤدي استخدام نفس الخطوط والألوان في واجهة المستخدم بأكملها إلى إنشاء مظهر ونمط عرض متناسقين، يوفر .NET MAUI طريقة لتعريف هذه القيم في مكان واحد، والبحث عنها في كل مكان يتم استخدامها فيه، يضمن إعادة استخدام القيم الاتساق في جميع أنحاء التطبيق، وإضفاء البساطة على التحديثات.

لنفترض أنك تقوم ببناء تطبيق للهواتف يسمى TipCalculator يُستخدم هذا التطبيق في مجال السياحة والفنادق ويسمح للمرافقين بحساب إكراميات أي خدمة بسرعة. لقد غيَّرت شركتك مؤخراً شكل علامتها التجارية، مهمتك هي تحديث واجهة مستخدم التطبيق لتتناسب مع المظهر الجديد، تحتاج إلى تغيير الخطوط وألوان النص وألوان الخلفية، تريد تسهيل هذا النوع من التحديثات لأنه من المؤكد أنه سيكون هناك المزيد من التغييرات في العلامة التجارية مع نمو شركتك.

في هذه الوحدة، ستتعلم كيفية تعريف وتطبيق مورد سواء في التعليمات البرمجية أو في XAML كما ستتعرف على كيفية تجميع إعدادات متعددة في نمط واحد، حتى تتمكن من تطبيقها جميعًا مرة واحدة.

٢ تحديد الموارد واستخدامها Define and use resources

المورد resource هو ثابت رمزي symbolic constant من لغة برمجة. ويمكنك تحديده في مكان واحد والإشارة إليه في كل مكان تحتاج إليه فيه، من الأسهل قراءة تعليماتك البرمجية لأنك تستخدم اسماً وصفياً بدلاً من قيمة `"magic" value` إذا احتجت إلى تغيير القيمة، فعندئذٍ يتعين عليك تحديث التعريف فقط.

في هذه الوحدة، سترى كيفية استخدام الموارد resources لإزالة القيم ذات التعليمات البرمجية المضمنة eliminate hard-coded values من XAML

ما هو المورد resource

المورد هو أي كائن يمكن مشاركته عبر واجهة المستخدم، ومن الأمثلة الأكثر شيوعاً عليه الخطوط والألوان والأحجام، ومع ذلك، يمكنك أيضاً تخزين كائنات معقدة complex objects مثل مثيلات Style and OnPlatform instances كموارد.

يمكنك تعيين مورد في XAML أو التعليمات البرمجية، بعد ذلك، يمكنك تطبيقه في XAML أو التعليمات البرمجية. عادة ما تعمل بالكامل في XAML على الرغم من أننا نعرض لك بعض الحالات لاحقاً حيث تكون التعليمات البرمجية مفيدة.

ضع في اعتبارك مثالاً، لنفترض أنك تريد استخدام نفس قيم TextColor عبر عناصر التحكم في صفحة، إذا استخدمت قيم تعليمات برمجية مضمَّنة، ستبدو XAML كما يلي، لاحظ كيف يتم تكرار قيمة لون النص TextColor في عنصري التحكم.

```
<Label TextColor="Blue" FontSize="14">
<Button TextColor="Blue" FontSize="14">
```

بدلاً من تكرار لون النص TextColor يمكنك تحديده كمورد resource يبدو التعريف مثل XAML هذا:

```
<Color x:Key="PageControlTextColor">Blue</Color>
```

لاحظ أن العنصر المعرف يحتوي على خاصية **`x:Key`** التي تمنح المورد اسمًا resource a name يمكنك استخدام هذا المفتاح key للبحث عن المورد في XAML

قبل أن تتمكن من استخدام مورد resource يجب تخزينه في قاموس مورد resource dictionary

ما هي ResourceDictionary

ResourceDictionary هي فئة مكتبة `.NET MAUI library class` مخصصة للاستخدام مع موارد واجهة المستخدم UI resources وهي تمثل قاموساً

dictionary لذلك، تخزن أزواج المفاتيح/القيم key/value pairs يقتصر نوع type of the key على String بينما يمكن أن تكون value أي كائن object

تحتوي كل صفحة XAML في .NET MAUI على خاصية تسمى Resources يمكنها الاحتفاظ بكائن ResourceDictionary object الخاصية فارغة بشكل افتراضي، لذا تحتاج إلى إنشاء مثيل قاموس dictionary instance قبل أن تتمكن من استخدامها، توضح التعليمات البرمجية التالية كيفية إنشاء كائن ResourceDictionary وتعيينه إلى الخاصية Resources لـ ContentPage

```
<ContentPage.Resources>
    <ResourceDictionary>
        ...
    </ResourceDictionary>
</ContentPage.Resources>
```

يحتوي .NET MAUI XAML على صيغة ملائمة مدمجة built-in convenience syntax تنشئ مثيل القاموس dictionary instance تلقائيًا كلما بدأت في استخدام خاصية الموارد Resources property يمكن تبسيط المثال السابق إلى التعليمات التالية:

```
<ContentPage.Resources>
    ...
</ContentPage.Resources>
```

يمكن أن تحتوي كل صفحة في تطبيقك على قاموس dictionary خاص بها. يمكنك استخدام هذه القواميس الخاصة بالصفحة page-specific dictionaries لتخزين الموارد resources المستخدمة حصريًا على تلك الصفحة.

ملاحظة:

يمكن أن يكون لكل عنصر تحكم على صفحة أيضا قاموس مورد resource dictionary خاص به. على سبيل المثال، يمكنك إضافة دليل مورد resource directory إلى عنصر تحكم Label مثل هذا:

```
<Label Text="Hello, World!"
        <Label.Resources>
            ...
        </Label.Resources>
</Label>
```

بصرف النظر عن التخطيطات وطرق العرض layouts and views التي يمكنها الاحتفاظ بعناصر فرعية child elements فليس من الشائع القيام بذلك على مستوى عنصر التحكم the control level

إنشاء مورد Create a resource

لإنشاء مورد، يمكنك تعريفه داخل خاصية المورد لصفحة the Resources property for a page ينشئ المثال التالي مورد لون النص text-color resource الموضح سابقاً:

```
<ContentPage.Resources>
  <Color x:Key="PageControlTextColor">Blue</Color>
</ContentPage.Resources>
```

عند تحديد مفتاح key لموردك، اختر اسمًا يعكس الاستخدام وليس قيمة المورد. على سبيل المثال، لتعيين خلفية الملصق باللون الأحمر، لا تستخدم اللون الأحمر كمفتاح RedColor as the key بل استخدم BackgroundColor بدلاً من ذلك.

تطبيق مورد Apply a resource باستخدام StaticResource

StaticResource هو ملحق علامات markup extension للبحث عن الموارد في قاموس الموارد resource dictionary يمكنك توفير مفتاح المورد key of the resource ثم تُرجِع علامة الامتداد القيمة المقابلة. تعرض XAML mark-up التالية مثالاً ينشئ ويستخدم `Color resource` يسمى PageControlTextColor تستخدم علامة XAML mark-up لعنصر label في المثال ملحق علامات StaticResource markup extension لاسترداد القيمة.

```
<ContentPage.Resources>
  <Color x:Key="PageControlTextColor">Blue</Color>
</ContentPage.Resources>

...

<Label TextColor="{StaticResource
PageControlTextColor}" ... />
```

يُطلق على الملحق The extension اسم StaticResource لأن الملحق يتم تقييمه مرة واحدة فقط. يحدث البحث في القاموس The dictionary عند إنشاء الكائن المستهدف. لا يتم تحديث خاصية الهدف إذا تغيرت قيمة المورد the resource value في القاموس.

تحذير:

يطرح StaticResource استثناء وقت التشغيل إذا لم يتم العثور على المفتاح the key

أنواع XAML المضمَّنة XAML intrinsic types

يعين المثال الأصلي المقدم في بداية هذا الدرس الخاصية TextColor وخاصية FontSize

```
<Label TextColor="Blue" FontSize="14">
<Button TextColor="Blue" FontSize="14">
```

يحتوي FontSize على النوع Double لإنشاء مورد لهذه القيمة، يمكنك استخدام أحد الأنواع المضمنة the XAML intrinsic types المحددة في مواصفات XAML تحدد مواصفات XAML أسماء الأنواع للعديد من أنواع C# البسيطة، تعرض التعليمات البرمجية التالية أمثلة على الموارد لكل نوع من الأنواع المضمنة.

```
<ContentPage.Resources>
    <x:String x:Key="...">Hello</x:String>
    <x:Char x:Key="...">X</x:Char>
    <x:Single x:Key="...">31.4</x:Single>
    <x:Double x:Key="...">27.1</x:Double>
    <x:Byte x:Key="...">8</x:Byte>
    <x:Int16 x:Key="...">16</x:Int16>
    <x:Int32 x:Key="...">32</x:Int32>
    <x:Int64 x:Key="...">64</x:Int64>
    <x:Decimal x:Key="...">12345</x:Decimal>
    <x:TimeSpan x:Key="...">1.23:5959</x:TimeSpan>
    <x:Boolean x:Key="...">True</x:Boolean>
</ContentPage.Resources>
```

Set platform-specific values for a resource تعيين قيم لمورد خاصة بنظام تشغيل

من الشائع إدخال تعديلات طفيفة على واجهة المستخدم بين أنظمة التشغيل، الطريقة القياسية لتعريف القيم الخاصة بالنظام هي باستخدام كائن OnPlatform object عند تعريف مورد. على سبيل المثال، توضح التعليمات البرمجية التالية كيفية إنشاء مورد يشير إلى ألوان نص text colors مختلفة على أنظمة iOS, Android, macOS (Mac Catalyst), Windows (WinUI)

```
<ContentPage.Resources>
 <OnPlatform x:Key="textColor" x:TypeArguments="Color">
     <On Platform="iOS" Value="Silver" />
     <On Platform="Android" Value="Green" />
     <On Platform="WinUI" Value="Yellow" />
     <On Platform="MacCatalyst" Value="Pink" />
 </OnPlatform>
</ContentPage.Resources>
...

<Label TextColor="{StaticResource textColor}" ... />
```

اختبار بسيط:

١- يتم تعيين القيمة التي يشير إليها `StaticResource` إلى متغير. ماذا يحدث عندما تتغير القيمة بعد ذلك؟

حل الاختبار:

١- لا شيء، يحدث البحث مرة واحدة فقط ولا تنعكس أي تغييرات أخرى في القاموس the backing dictionary

يتم تقييم StaticResource مرة واحدة فقط عند تعيين قيمة الخاصية الهدف the target property value

٣ تمرين: استخدام الموارد على مستوى الصفحة page-level resources

ترتبط جميع التمرينات في هذه الوحدة بتطبيق TipCalculator المُنشأ مسبقاً. يمكنك تعديل هذا التطبيق وتحسينه خلال الوحدة، في هذا التمرين، يمكنك استخدام موارد على مستوى الصفحة لإزالة قيم XAML المتكررة.

تستخدم هذه الوحدة .NET 9.0 SDK. تأكد من تثبيت .NET 9.0 عن طريق تشغيل الأمر التالي في موجة الأوامر command terminal

```
dotnet --list-sdks
```

يظهر إخراج مشابه للمثال التالي:

```
8.0.100 [C:\Program Files\dotnet\sdk]
9.0.100 [C:\Program Files\dotnet\sdk]
```

تأكد من إدراج إصدار يبدأ بـ 9 إذا لم يتم سرد أي منها أو لم يتم العثور على الأمر، فقم بتثبيت أحدث إصدار .NET 9.0 SDK

أفتح the starter solution

١- نسخ أو تنزيل exercise repo من GitHub

ملاحظة:

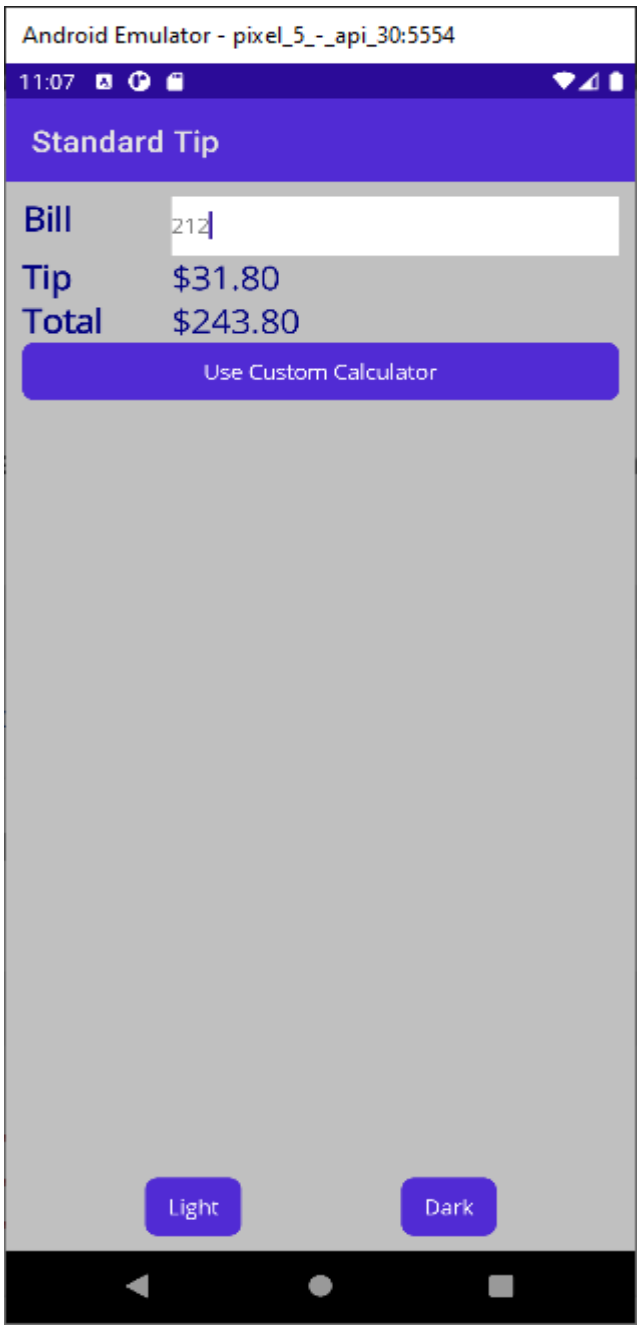
من الأفضل نسخ محتوى التمرين أو تنزيله إلى مسار قصير، مثل C:\dev لتجنب تجاوز الملفات المنشأة الحد الأقصى لطول المسار.

٢- فتح the starter solution من المجلد exercise1/TipCalculator باستخدام Visual Studio أو هذا المجلد مباشرة في Visual Studio Code

٣- تحقق من إنشائه وتشغيله builds and runs في بيئتك (لا بأس باستخدام أي نظام)

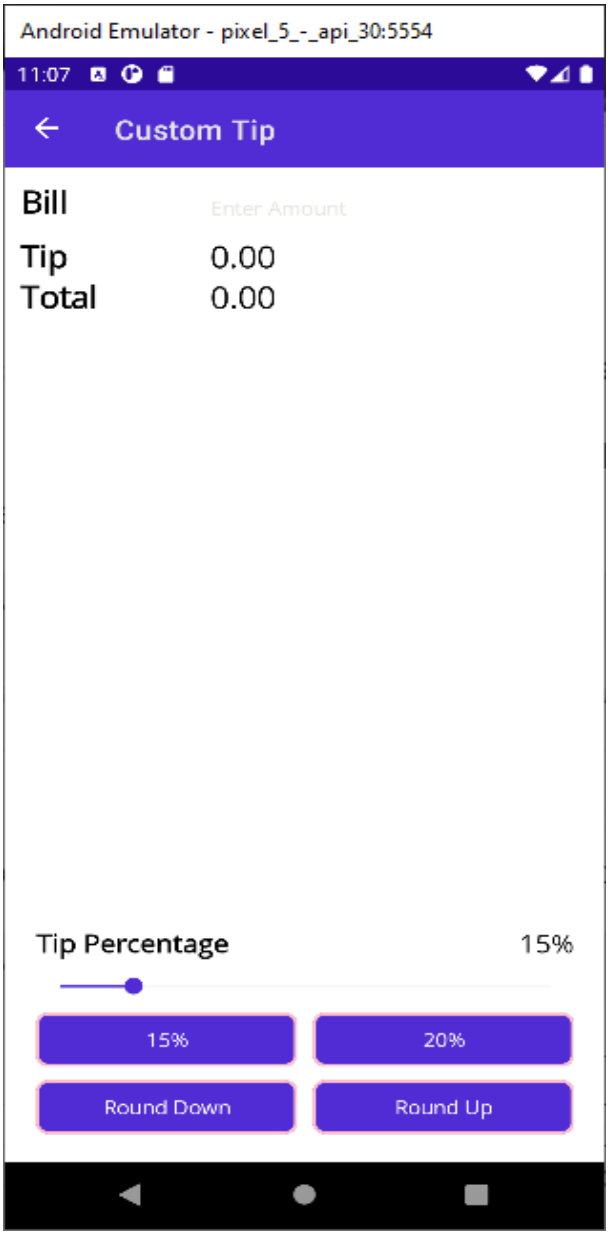
٤- خذ بضع دقائق لفحص التطبيق وتشغيله حتى تفهم كيفية عمله.

٥- يوفر التطبيق صفحتين، صفحة StandardTipPage page هي حاسبة تلميح tip calculator بسيطة، تقوم بإدخال قيمة، وتحسب الصفحة الإكرامية tip (15%) والإجمالي total المستحق. تظهر الصورة التالية التطبيق الذي يعمل على جهاز Android



تتيح لك الأزرار Light and Dark buttons تغيير نسق لون الصفحة، الإعداد الافتراضي هو النسق الفاتح the Light theme إذا قمت بتحديد Dark يتم عكس ألوان الخلفية والنص the colors for the background and text

يقوم زر Use Custom Calculator button بتبديل العرض إلى صفحة the CustomTipPage page تتيح لك هذه الصفحة تغيير نسبة المئوية للإكرامية tip باستخدام شريط تمرير slider يمكنك أيضًا تحديد الزرين 15% and 20% buttons لحساب tip استنادًا إلى معدلات محددة مسبقًا.



البحث عن XAML المتكررة

١- افتح ملف StandardTipPage.xaml

٢- ابحث عن علامات XAML markup التي تعين لون خلفية background color لـ LayoutRoot grid لاحظ أنه يستخدم قيمة تعليمات برمجية مضمنة hard-coded value

```
<Grid x:Name ="LayoutRoot"
BackgroundColor="Silver" Padding="10">
```

٣- ابحث عن XAML markup الذي يعين لون نص الملصقات text color of the labels في العمود الأيسر "left column" إلى Navy وحجم الخط إلى font size to 22 لاحظ أنه يتم استخدام نفس القيم في ثلاث labels

```
<!-- Left column = static labels -->
<Label x:Name="billLabel"  Text="Bill"
TextColor="Navy" FontSize="22" ... />
<Label x:Name="tipLabel"   Text="Tip"
TextColor="Navy" FontSize="22" ... />
<Label      x:Name="totalLabel"      Text="Total"
TextColor="Navy" FontSize="22" ... />
```

٤- ابحث عن التعليمة XAML التي تعين لون color of the labels في العمود الأيمن "right column" إلى Navy وحجم النقطة إلى point size to 22 لاحظ أن نفس القيم تُستخدم لـ two labels

يبدو أن بعض إعدادات الخصائص تشكل مجموعة منطقية logical group على سبيل المثال، يتم استخدام تركيبة Navy and 22 على ملصقات متعددة multiple labels

```
<!-- Right column = user input and calculated-value output
-->
<Entry ... />
<Label x:Name="tipOutput"   Text="0.00"
TextColor="Navy" FontSize="22"   ... />
<Label     x:Name="totalOutput"     Text="0.00"
TextColor="Navy" FontSize="22"   ... />
```

فكر في الجهد المطلوب لتعديل قيم لون النص وحجم الخط TextColor and FontSize values ستحتاج إلى تغييرها في خمسة أماكن.

تحديد الموارد Define resources

الآن دعونا ننشئ موارد في XAML حتى تتمكن من البدء في إزالة بعض التعليمات البرمجية المتكررة التي وجدتها في التطبيق.

١- افتح ملف StandardTipPage.xaml

٢- تعريف مورد لون Color resource داخل قسم ContentPage.Resources امنح المورد x:Key resource المعرف bgColor والقيمة #C0C0C0 (يمكنك أيضا استخدام اسم اللون the name of the color Silver)

٣- تعريف مورد لون Color resource ثان، أعط المورد x:Key resource المعرف fgColor والقيمة #0000AD (يمكنك أيضا استخدام اسم اللون the name of the color Navy)

٤- تعريف المورد x:Double resource بمعرف fontSize تعيين قيمة هذا المورد إلى 22

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             ...>

  <ContentPage.Resources>
     <ResourceDictionary>
        <Color x:Key="bgColor">#C0C0C0</Color>
        <Color x:Key="fgColor">#0000AD</Color>
       <x:Double x:Key="fontSize">22</x:Double>
     </ResourceDictionary>
  </ContentPage.Resources>

  <Grid x:Name ="LayoutRoot" ...>
        ...
```

استخدام الموارد الثابتة static resources

فلنطبق الآن الموارد التي أنشأتها.

١- استخدم ملحق علامة StaticResource mark-up extension لتطبيق مورد the bgColor resource على الخاصية Background لعنصر تحكم LayoutRoot Grid

```
...
```

```
<Grid x:Name ="LayoutRoot"
BackgroundColor="{StaticResource bgColor}"
Padding="10">
```

٢- قم بتطبيق مورد fgColor على خاصية TextColor لكافة عناصر Label التي تقوم حاليًا بتعيين TextColor إلى Navy بالإضافة إلى ذلك، قم باستبدال حجم الخط المضمن او المبرمج مسبقًا hard-coded font size بالمورد الثابت fontSize static resource

```
...
<!-- Left column = static labels -->
<Label x:Name="billLabel"  Text="Bill"
TextColor="{StaticResource fgColor}"
FontSize="{StaticResource fontSize}" ... />
<Label x:Name="tipLabel"   Text="Tip"
TextColor="{StaticResource fgColor}"
FontSize="{StaticResource fontSize}" ... />
<Label x:Name="totalLabel" Text="Total"
TextColor="{StaticResource fgColor}"
FontSize="{StaticResource fontSize}" ... />
...
```

٣- قم بتشغيل التطبيق. تأكد من أن StandardTipPage لا يزال يعرض نصًا داكنًا على خلفية فاتحة عند بدء التشغيل، كما كان يفعل من قبل.

ملاحظة:

في هذه المرحلة؛ لا تقلق بشأن تصميم CustomTipPage أو النسق الفاتح والداكن the Light and Dark themes ستعالج هذه المشكلات لاحقاً.

٤ استخدام الموارد التفاعلية وتحديثها update dynamic resources

في الدرس السابق، حددتَ مورداً defined a resource في XAML واستخدمته كقيمة ثابتة static value ومع ذلك، هناك بعض الحالات التي لا يكون فيها StaticResource مناسباً، ضع في الاعتبار السيناريوهات التالية:

- لنفترض أنك تريد تنفيذ نسق الألوان color themes التي تسمح للمستخدم بتغيير مظهر التطبيق في وقت التشغيل، يقوم ملحق علامة StaticResource mark-up extension بالبحث عن القاموس the dictionary مرة واحدة فقط، لذلك لا يمكنه تحديث واجهة المستخدم بالتفاعل UI dynamically
- يمكنك تخزين تفضيلات المستخدم على خادم ويب وتحميلها عند بدء تشغيل التطبيق، يطرح ملحق علامة StaticResource استثناء إذا لم يتم العثور على المفتاح key في القاموس the dictionary

يوضح هذا الدرس كيفية معالجة مشكلات مثل هذه باستخدام الموارد التفاعلية dynamic resources

كيفية تحديث الموارد في وقت التشغيل resources at runtime

يمكنك تخزين الموارد في قاموس مورد resource dictionary يمكنك كتابة التعليمات البرمجية لتحديث هذه الموارد في وقت التشغيل، يمكنك أيضاً إضافة موارد جديدة أو إزالة الموارد الموجودة.

تأمل المثال التالي:

```
<ContentPage.Resources>
  <Color x:Key="PanelBackgroundColor">Blue</Color>
</ContentPage.Resources>
```

لنفترض أنك تريد تغيير قيمة مورد PanelBackgroundColor resource أثناء تشغيل التطبيق. يمكنك إضافة تعليمة إلى ملف التعليمات البرمجية الخلفي للصفحة، للوصول إلى الخاصية Resources يقوم المثال التالي بتحديث قيمة المورد من مثال XAML السابق إلى لون مختلف.

```
this.Resources["PanelBackgroundColor"] = Colors.Green;
```

ما هو DynamicResource

DynamicResource هو ملحق آخر للعلامات، للبحث عن الموارد في قاموس الموارد resource dictionary وهو مشابه لـ StaticResource من حيث أنه يقوم بالبحث في القاموس عند إنشاء الكائن الهدف the target object لكنه يستمع أيضاً

للتغييرات التي تطرأ على المورد الموجود في القاموس. إذا تغيرت قيمة المورد في القاموس، يقوم DynamicResource بتحديث واجهة المستخدم تلقائيًا.

DynamicResource له ميزة عن StaticResource إذا تعذر على DynamicResource العثور على المفتاح key في القاموس، فإنه يترك الخاصية بدون تعيين، على عكس StaticResource فإن المفتاح key المفقود ليس خطأً ولا يطرح استثناءً.

ملاحظة:

تفرض طبيعة the DynamicResource mark-up extension عقوبة أداء صغيرة على تطبيقك، على الرغم من أنه يمكنك استخدام DynamicResource بدلاً من StaticResource في صفحات XAML pages إذا لم يتغير المورد، فيجب عليك الرجوع إليه مع ملحق علامة StaticResource mark-up extension

لاستخدام لون الخلفية background color المحدث من المثال السابق، يمكنك تطبيق DynamicResource في التعليمات البرمجية XAML مثل هذا:

```
<ContentPage ...>
 <ContentPage.Resources>
  <Color x:Key="PanelBackgroundColor">Blue</Color>
 </ContentPage.Resources>

  <StackLayout BackgroundColor="{DynamicResource
PanelBackgroundColor}">
    ...
    </StackLayout>
</ContentPage>
```

إذا تغيرت قيمة PanelBackgroundColor resource فسيتم تحديث BackgroundColor value لعنصر StackLayout تلقائيًا.

اختبار بسيط:

١- ما هي الميزة التي يوفرها DynamicResource ولا يوفرها StaticResource

حل الاختبار:

١- يستمع إلى التغييرات في القاموس الداعم the backing Dictionary ويحدث واجهة المستخدم عند حدوث أي تغيير.

DynamicResource هو ملحق للبحث عن القيم في قاموس، ولكنه ينشر أي تغييرات في تلك القيم إلى واجهة المستخدم.

٥ تمرين: استخدام الموارد التفاعلية dynamic resources لتحديث العناصر update elements

في هذا التمرين، يمكنك استخدام ملحق علامة DynamicResource لتحديث واجهة مستخدم TipCalculator عند تغيير قيم الموارد resource values

يُعد هذا التمرين استمراراً للتمرين السابق، إما أن تستخدم الحل الموجود كنقطة بداية لهذه الخطوات، أو افتح مشروع TipCalculator في المجلد exercise2/TipCalculator في المستودع الذي نسخته في التمرين السابق.

البحث عن تعليمات برمجية متكررة

ينفذ التطبيق أنظمة ألوان بسيطة فاتحة وداكنة "light" and "dark" color لصفحة StandardTipPage يمكنك هنا فحص التعليمات البرمجية المستخدمة لتغيير الألوان.

١- افتح ملف StandardTipPage.xaml.cs

حدد موقع معالجي الأحداث event handlers اللذين يحدِّثان ألوان واجهة المستخدم the UI colors

```
private Color colorNavy = Colors.Navy;
private Color colorSilver = Colors.Silver;

...

void OnLight(object sender, EventArgs e)
{
    LayoutRoot.BackgroundColor = colorSilver;

    tipLabel.TextColor = colorNavy;
    billLabel.TextColor = colorNavy;
    totalLabel.TextColor = colorNavy;
    tipOutput.TextColor = colorNavy;
    totalOutput.TextColor = colorNavy;
}

void OnDark(object sender, EventArgs e)
{
```

```
    LayoutRoot.BackgroundColor = colorNavy;

    tipLabel.TextColor = colorSilver;
    billLabel.TextColor = colorSilver;
    totalLabel.TextColor = colorSilver;
    tipOutput.TextColor = colorSilver;
    totalOutput.TextColor = colorSilver;
}
```

لاحظ كيف تقوم التعليمات البرمجية بتحديث الألوان لكل عنصر تحكم على حدة، ما يؤدي إلى تكرار التعليمات البرمجية.

تحديث الموارد من التعليمات البرمجية Update resources from code

تبدأ بكتابة التعليمات البرمجية التي تحدث بعض الموارد المخزنة resources stored في قاموس موارد الصفحة page's resource dictionary

١- إزال كافة التعليمات البرمجية من أسلوب `OnLight method`

٢- أضف التعليمات البرمجية التالية الموضحة إلى `OnLight method` تقوم هذه التعليمة البرمجية بتعيين مورد fgColor resource في قاموس موارد الصفحة إلى القيمة في متغير colorNavy variable وتعيين مورد bgColor resource إلى القيمة في متغير colorSilver variable

تستخدم المتغيرات colorNavy and colorSilver الأسلوب الثابت static Color.FromRgb method ما يجعل من السهل تحويل قيمة سداسية عشرية إلى لون hexadecimal value to a color

```
void OnLight(object sender, EventArgs e)
{
    Resources["fgColor"] = colorNavy;
    Resources["bgColor"] = colorSilver;
}
```

٣- كرر الخطوتين السابقتين مع الأسلوب الثاني `OnDark method` ولكن قم بعكس الألوان؛ قم بتعيين fgColor إلى colorSilver و bgColor إلى colorNavy

```
void OnDark(object sender, EventArgs e)
{
```

```
    Resources["fgColor"] = colorSilver;
    Resources["bgColor"] = colorNavy;
}
```

٤- قم بتشغيل التطبيق. حدد الزرين Dark and Light buttons لا تتغير واجهة المستخدم، على الرغم من أن التعليمات تغير قيم الموارد في القاموس the resource values in the dictionary إلا أن القيمة الجديدة لا يتم نشرها إلى واجهة المستخدم. تكمن المشكلة في أنك تستخدم ملحق علامات StaticResource mark-up extension لتعيين القيم في تعليمات XAML

تحديث واجهة المستخدم بالتفاعل Update the UI dynamically

لإصلاح المشكلة، قم بتعديل XAML بحيث يتم تحميل قيم الموارد المحدثة في واجهة المستخدم الخاصة بك.

١- أوقف التطبيق، وافتح الملف StandardTipPage.xaml

٢- حدد موقع جميع الأماكن التي تعيّن فيها الألوان من قيم الموارد، استبدل استخدام ملحق علامة StaticResource ب DynamicResource كما هو موضح في هذا المثال.

```
 <Grid x:Name ="LayoutRoot"
BackgroundColor="{DynamicResource bgColor}"
Padding="10">
...
 <Label x:Name="billLabel"  Text="Bill"
TextColor="{DynamicResource fgColor}" ... />
 <Label x:Name="tipLabel"   Text="Tip"
TextColor="{DynamicResource fgColor}" ... />
 <Label x:Name="totalLabel" Text="Total"
TextColor="{DynamicResource fgColor}" ... />
...
```

ملحوظة: لا تقم بتغيير خصائص FontSize من StaticResource إلى DynamicResource

٣- تشغيل التطبيق. حدد الزرين Dark and Light يتم الآن تحديث واجهة المستخدم بشكل صحيح.

٦ إنشاء واجهة مستخدم متسقة consistent UI باستخدام الأنماط styles

الموارد Resources مفيدة لتجنب القيم المكررة ذات التعليمات البرمجية المضمنة في علامة XAML ولكن ربما يكون تطبيقها مرهقًا، يمكنك تعيين قيمة كل خاصية على حدة، مما قد يؤدي إلى إنشاء XAML مزدحم ومطول، يوضح لك هذا الدرس كيفية تجميع إعدادات متعددة multiple settings في نمط style مما قد يساعد في إزالة الفوضى من تعليماتك، وجعلها أكثر قابلية للصيانة.

كيف يمكن للموارد أن تشوش resources can clutter على XAML

يوفر المورد قيمة لخاصية واحدة، ومع ذلك، فإن استخدام الكثير من الموارد قد يؤدي إلى XAML مطوّل، لنفترض أنك تريد مظهرًا مخصصًا لأزرارك. قم أولاً بإنشاء موارد للقيم التي تحتاجها، ثم قم بتطبيق كل مورد على جميع الأزرار، توضح التعليمات التالية كيف قد يبدو XAML mark-up لزرين.

```
<Button
 Text = "OK"
 BackgroundColor = "{StaticResource highlightColor}"
 BorderColor = "{StaticResource borderColor}"
 BorderWidth = "{StaticResource borderWidth}"
 TextColor = "{StaticResource textColor}" />

<Button
 Text = "Cancel"
 BackgroundColor = "{StaticResource highlightColor}"
 BorderColor = "{StaticResource borderColor}"
 BorderWidth = "{StaticResource borderWidth}"
 TextColor = "{StaticResource textColor}" />
```

لاحظ كيفية تعيين نفس الخصائص الخمسة في كل زر من الأزرار، يؤدي استخدام الموارد إلى إزالة الحاجة إلى تكرار القيم ذات تعليمات برمجية مضمنة في أربع منها. ومع ذلك، يصعب قراءة هذا النوع من علامات XAML بسرعة، بالإضافة إلى ذلك، إذا كنت تقوم بتعيين عدد كبير من الخصائص لكل عنصر تحكم، فمن السهل حذف أحدها عن طريق الخطأ، ما يؤدي إلى عدم تناسق في مظهر عناصر التحكم، الحل هو إنشاء نمط يعين جميع الخصائص الأربعة دفعة واحدة.

ما هي setter

تترجم إلى محدد أو معين، Setters هي المكونات الأساسية التي تستخدمها لإنشاء الأنماط styles

setter هي حاوية لزوج مكون من خاصية/قيمة property/value يمكنك التفكير في setter على أنها تمثل عبارة تعيين assignment statement يمكنك تحديد الخاصية التي يجب تعيينها والقيمة التي يجب تطبيقها، عادةً ما تقوم بإنشاء كائنات Setter في علامة XAML ينشئ المثال التالي Setter object للخاصية TextColor

```
<Setter Property="TextColor" Value="White" />
```

يمكنك استخدام مورد للقيمة resource for the value في محدد setter كما هو موضح في التعليمات التالية، هذه التقنية رائعة عندما تريد استخدام نفس القيمة في محددات متعددة multiple setters

```
<Setter Property="TextColor"
Value="{StaticResource textColor}" />
```

ملاحظة:

يجب تنفيذ قيمة الخاصية التي تحددها في setter كخاصية قابلة للربط bindable property جميع الخصائص على عناصر التحكم في .NET MAUI. التي تنتهي بخاصية اللاحقة suffix Property هي خصائص قابلة للربط bindable properties إذا كنت تحاول استخدام خاصية مثل TextColor في setter فتأكد من وجود خاصية قابلة للربط مطابقة تسمى TextColorProperty لعنصر التحكم هذا، عملياً، تنفذ جميع الخصائص التي تريد استخدامها في setters تقريباً بهذه الطريقة.

ما هو النمط style

النمط style هو مجموعة من المعينات setters التي تستهدف نوعًا معيناً من عناصر التحكم specific type of control يتطلب .NET MAUI. نوعًا مستهدفًا target type حتى يتمكن من التأكد من وجود الخصائص في setters الخاصة بك على هذا النوع.

تُظهر التعليمات البرمجية التالية نمطاً يجمع بين القيم الأربع من المثال السابق، لاحظ أنه تم تعيين TargetType إلى Button وجميع الخصائص في setters أعضاء في فئة Button class لا يمكنك استخدام هذا النمط ل Label لأن فئة Label class لا تتضمن الخاصية BorderColor أو BorderWidth

```
<Style TargetType="Button">
<Setter Property="BackgroundColor" Value="#2A84D3" />
 <Setter Property="BorderColor" Value="#1C5F9B" />
 <Setter Property="BorderWidth" Value="3" />
 <Setter Property="TextColor" Value="White" />
</Style>
```

تحديد نمط Define a style

عادةً ما تقوم بتعريف الأنماط كموارد داخل ResourceDictionary object يسهل قاموس الموارد resource dictionary استخدام النمط style عبر عناصر تحكم متعددة في نفس الصفحة، أو حتى عبر تطبيقك بالكامل. توضح التعليمات التالية كيفية تعريف نمط كمورد داخل قاموس، لاحظ أن النمط مُعطى اسمًا باستخدام الخاصية x:Key يتيح لك تسمية النمط الرجوع إليه من داخل صفحات XAML pages

```
<ContentPage.Resources>
 <Style x:Key="MyButtonStyle" TargetType="Button">
        ...
  </Style>
</ContentPage.Resources>
```

تطبيق نمط Apply a style

يمكنك إرفاق نمط بعنصر تحكم عن طريق تعيين الاسم إلى الخاصية Style property يؤدي التعيين إلى تطبيق كل كائن من كائنات Setter objects في النمط على عنصر التحكم المستهدف، توضح التعليمات التالية كيفية تطبيق نمط زر button style على زرين

```
<Button Text="OK" Style="{StaticResource
MyButtonStyle}" />
<Button  Text="Cancel"  Style="{StaticResource
MyButtonStyle}" />
```

في المثال السابق، استخدمت ملحق علامات StaticResource mark-up extension لربط النمط attach the style بعناصر التحكم، هذه التقنية رائعة عندما لا تحتاج إلى تغيير النمط في وقت التشغيل، ولكن ماذا لو كنت تريد تنفيذ شيء مثل السمات الحيوية أو التفاعلية dynamic themes حيث تحتاج واجهة المستخدم إلى التغيير؟ في هذه الحالة، يمكنك استخدام ملحق علامات DynamicResource mark-up extension لتحميل النمط load the style

```
<Button  Text="Cancel"  Style="{DynamicResource
MyButtonStyle}" />
```

يستجيب DynamicResource لاستبدال الخاصية **x:Key property** في قاموس المورد resource dictionary إذا كتبت تعليمة برمجية تقوم بتحميل نمط جديد في ResourceDictionary بنفس قيمة x:Key value يتم تطبيق النمط الجديد تلقائياً على واجهة المستخدم الخاصة بك.

استخدام نمط ضمني implicit style لعناصر تحكم متعددة multiple controls

لنفترض أن واجهة المستخدم تحتوي على ٥٠ زراً وأنك تريد تطبيق نفس النمط على جميع الأزرار، بناءً على ما نعرفه حتى الآن، ستحتاج إلى تعيين خاصية Style على كل زر يدويًا، ليس من الصعب إجراء ذلك، لكنه يظل أمراً مرهقًا.

النمط الضمني implicit style هو نمط تضيفه إلى قاموس مورد resource dictionary دون إعطائه معرف x:Key identifier يتم تطبيق الأنماط الضمنية تلقائياً على كافة عناصر التحكم في كائن TargetType object المحدد.

تظهر التعليمات التالية المثال السابق المعلن كنمط ضمني implicit style يتم تطبيق هذا النمط على كل زر على الصفحة.

```
<ContentPage.Resources>
  <Style TargetType="Button">
   <Setter Property="BackgroundColor" Value="Blue" />
    <Setter Property="BorderColor" Value="Navy" />
      ...
  </Style>
</ContentPage.Resources>
```

هام: تتطلب مطابقة الأنماط الضمنية لعناصر التحكم تطابقًا دقيقًا مع TargetType المحدد، لن تستطيع عناصر التحكم التي ترث من نوع الهدف تلقي الأنماط، للتأثير على عناصر التحكم الموروثة، يمكنك تعيين سمة Style.ApplyToDerivedTypes attribute على True عند تعريف النمط. على سبيل المثال، لتطبيق نمط على نوع الزر وجعله يؤثر على أي من الأزرار التي ترث من الزر (مثل ImageButton أو RadioButton أو نوع مخصص تقوم بإنشائه) يمكنك استخدام نمط مثل هذا.

```
<ContentPage.Resources>
 <Style TargetType="Button"
          ApplyToDerivedTypes="True">
  <Setter Property="BackgroundColor" Value="Black" />
```

```
 </Style>
</ContentPage.Resources>
```

تجاوز نمط Override a style

يمكنك التفكير في النمط على أنه يوفر مجموعة من القيم الافتراضية لعناصر التحكم. قد يكون النمط الموجود قريباً من متطلباتك، ولكنه يحتوي على معين setters واحد أو اثنين لا تريدهما، في هذه الحالة، يمكنك تطبيق النمط ثم تجاوز القيمة override the value عن طريق تعيين الخصائص مباشرة، يتم تطبيق الإعداد الصريح بعد النمط، وبالتالي سيتم تجاوز القيمة من النمط.

لنفترض أنك تريد استخدام النمط التالي لعدة أزرار على صفحتك.

```
<Style x:Key="MyButtonStyle"
TargetType="Button">
  <Setter Property="BackgroundColor"
Value="Blue" />
  <Setter Property="BorderRadius" Value="10" />
  <Setter Property="BorderWidth" Value="3" />
</Style>
```

يعمل هذا النمط مع جميع الأزرار باستثناء زر Cancel الذي يحتاج إلى خلفية حمراء. يمكنك استخدام نفس النمط لزر Cancel طالما قمت أيضًا بتعيين الخاصية BackgroundColor مباشرة، توضح التعليمات التالية كيفية تجاوز إعداد اللون.

```
<Button
    Text="Cancel"
    Style="{StaticResource MyButtonStyle}"
    BackgroundColor="Red"
    ... />
```

استهداف نوع عنصر أصل Target an ancestor type

لنفترض أنك تريد لون خلفية مخصصاً custom background color لكل من buttons and labels يمكنك إنشاء أنماط منفصلة لكل نوع، أو يمكنك إنشاء نمط واحد مع تعيين TargetType إلى VisualElement تعمل هذه التقنية لأن VisualElement هي فئة أساسية base class لكل من Button and Label

توضح التعليمات التالية نمطاً يستهدف فئة أساسية يتم تطبيقها على نوعين مشتقين مختلفين.

```
<Style x:Key="MyVisualElementStyle"
TargetType="VisualElement">
 <Setter Property="BackgroundColor" Value="#2A84D3"
/>
</Style>

<Button Style="{StaticResource MyVisualElementStyle}"
... />
<Label Style="{StaticResource MyVisualElementStyle}"
... />
```

يحدد هذا المثال النمط style باستخدام x:Key وتطبقه عناصر التحكم بشكل صريح. لا يعمل النمط الضمني implicit style هنا لأن TargetType للنمط الضمني يجب أن يكون مطابقاً تماماً لنوع عنصر التحكم.

استخدم BasedOn للوراثة من نمط inherit from a style

لنفترض أنك تريد إنشاء مظهر متماسك لواجهة المستخدم، فتقرر أن جميع عناصر التحكم يجب أن تستخدم لون خلفية متناسقًا، من المرجح أن يظهر إعداد لون الخلفية في أكثر من نمط من أنماطك. توضح التعليمات التالية نمطين two styles مع repeated setter

```
<Style x:Key="MyButtonStyle" TargetType="Button">
 <Setter Property="BackgroundColor" Value="Blue" />
  <Setter Property="BorderColor" Value="Navy" />
  <Setter Property="BorderWidth" Value="5" />
</Style>

<Style x:Key="MyEntryStyle" TargetType="Entry">
 <Setter Property="BackgroundColor" Value="Blue" />
```

```
  <Setter Property="TextColor" Value="White" />
</Style>
```

يمكنك استخدام وراثة النمط style inheritance لتقسيم setter المكرر إلى نمط أساسي base style لإنشاء نمط مشتق derived style قم بتعيين الخاصية BasedOn الخاصة به للإشارة إلى النمط الأساسي. يرث النمط الجديد جميع the setters من نمطه الأساسي base style يمكن للنمط المشتق derived style أيضاً إضافة setters جديدة أو استبدال معين موروث inherited setter بنمط يحتوي على قيمة مختلفة.

توضح التعليمات التالية أنماط المثال السابق المعاد صياغتها في تدرج هرمي، لا يظهر setter الشائع إلا في النمط الأساسي بدلاً من تكراره، لاحظ أنك تستخدم ملحق علامة StaticResource للبحث عن النمط الأساسي. لا يمكنك استخدام DynamicResource في هذه الحالة.

```
<Style x:Key="MyVisualElementStyle"
TargetType="VisualElement">
    <Setter Property="BackgroundColor"
Value="Blue" />
</Style>

<Style x:Key="MyButtonStyle" TargetType="Button"
BasedOn="{StaticResource MyVisualElementStyle}">
    <Setter Property="BorderColor" Value="Navy" />
    <Setter Property="BorderWidth" Value="5" />
</Style>

<Style x:Key="MyEntryStyle" TargetType="Entry"
BasedOn="{StaticResource MyVisualElementStyle}">
    <Setter Property="TextColor" Value="White" />
</Style>
```

يجب أن تكون قيمة TargetType value للأنماط الأساسية والمشتقة متوافقة، لكي تكون الأنماط متوافقة، يجب أن يكون لهم نفس خاصية TargetType property أو يكون TargetType للنمط المشتق من سلالة TargetType للنمط الأساسي descendent of the TargetType of the base style

اختبار بسيط:

١- لماذا تقوم بتعريف نمط style داخل قاموس الموارد ResourceDictionary

حل الاختبار:

١- يسهل قاموس الموارد استخدام النمط عبر عناصر تحكم متعددة في نفس الصفحة.

يمكنك الوصول إلى القيم access the values بسهولة عن طريق الإشارة إلى مفتاح قاموس الموارد referring to the resource dictionary's key

٧ تمرين: إنشاء نمط وتطبيقه Create and apply a style

في هذا التمرين، يمكنك تعريف نمط على مستوى الصفحة وتطبيقه define and apply a page-level style في تطبيق Tip Calculator

هذا التمرين هو استمرار للتمرين السابق، إما أن تستخدم الحل الحالي الخاص بك كنقطة بداية لهذه الخطوات، أو افتح مشروع TipCalculator في المجلد exercise3/TipCalculator في المستودع الذي قمت باستنساخه في التمرين الأول.

تحديد نمط Define a style

لنبدأ بتنفيذ نمط خط `"size 22 Bold" font style` لاستخدامه labels وتخزين النمط Store the style في قاموس على مستوى الصفحة page-level dictionary

١- في مشروع TipCalculator افتح الملف StandardTipPage.xaml

٢- أضف نمطاً Style إلى قاموس موارد الصفحة the page's resource dictionary بعد الموارد الموجودة، استخدم قيمة x:Key لـ infoLabelStyle وقيمة TargetType لـ Label

```
<ContentPage.Resources>
 <ResourceDictionary>
    ...
 <Style x:Key="infoLabelStyle" TargetType="Label">
  </Style>
 </ResourceDictionary>
</ContentPage.Resources>
```

٣- أضف مثيل Setter instance الذي يعين الخاصية FontSize للنمط style إلى القيمة fontSize في مورد resource

٤- أضف Setter آخر يقوم بتعيين الخاصية FontAttributes إلى Bold

```
<Style x:Key="infoLabelStyle" TargetType="Label">
 <Setter Property="FontSize"
Value="{StaticResource fontSize}" />
 <Setter Property="FontAttributes" Value="Bold" />
</Style>
```

تطبيق النمط Apply the style

١- حدد موقع عناصر تحكم Label الثلاث التي تستخدم قيمة FontSize لـ {StaticResource fontSize} وقيمة FontAttributes بتنسيق Bold قم بإزالة تعيينات الخصائص هذه من labels

٢- استخدم ملحق علامة StaticResource لتعيين نمط the infoLabelStyle style لهذه labels الثلاث:

```
<!-- Left column = static labels -->
 <Label x:Name="billLabel"  Text="Bill"  ...
Style="{StaticResource infoLabelStyle}" ... />
 <Label x:Name="tipLabel"   Text="Tip"   ...
Style="{StaticResource infoLabelStyle}" ... />
 <Label   x:Name="totalLabel"   Text="Total"   ...
Style="{StaticResource infoLabelStyle}" ... />
```

٣- تشغيل التطبيق. يجب أن يبدو التطبيق كما كان من قبل تماماً. ومع ذلك، يتم الآن تعيين سمات الخط the font attributes لـ labels باستخدام نمط style

تغيير نمط خط Change the font style

دعونا نلقي نظرة على كيفية سهولة تحديث الأنماط.

١- ارجع إلى النمط style في قاموس الموارد the resource dictionary وغير the fontSize resource إلى 32

٢- شغّل التطبيق مرة أخرى لعرض التغييرات، يجب أن تكون labels الثلاث Bill, Tip, Total أكبر.

٣- قم بتغيير the fontSize resource مرة أخرى إلى 22

إنشاء نمط أساسي base style

دعونا نوسع تنفيذ صفحة the StandardTipPage page بإضافة نمط أساسي base style يمكنك تعريف نمط جديد بقيم تتداخل مع النمط الذي قمت بإنشائه في الخطوات السابقة، بعد ذلك، يمكنك إعادة بناء التعليمات البرمجية لهذا النمط الجديد في الجزء التالي من هذا التمرين.

١- افتح ملف StandardTipPage.xaml

٢- إضافة نمط آخر إلى قاموس موارد الصفحة the page's resource dictionary
استخدم قيمة x:Key لـ baseLabelStyle وقيمة TargetType لـ Label

هام:

حدد هذا النمط أعلى نمط infoLabelStyle سيكون هذا الوضع مهمًا لاحقًا عند الوراثة من هذا النمط، لا يمكن للنمط أن يرث إلا من نمط آخر موجود بالفعل في النطاق.

٣- أضف Setter الذي يعين الخاصية FontSize لاحظ أن هذا setter هو تكرار لـ setter من infoLabelStyle السابق.

```
<ContentPage.Resources>
 <ResourceDictionary>
    ...
 <Style x:Key="baseLabelStyle" TargetType="Label">
  <Setter Property="FontSize"
Value="{StaticResource fontSize}" />
  </Style>
    ...
 <ResourceDictionary>
</ContentPage.Resources>
```

٤- تطبيق baseLabelStyle الجديد على labels في الصفحة التي تعرض المبالغ المحسوبة للإكرامية والإجمالي the tip and the total قم بإزالة إعدادات FontSize الصريحة من هذه labels تبين التعليمات التالية مثالاً:

```
<!-- Right column = user input and calculated-
value output -->
...
<Label x:Name="tipOutput"   Text="0.00"
TextColor="Navy" Style="{StaticResource
baseLabelStyle}" Grid.Row="1" Grid.Column="1" />
<Label      x:Name="totalOutput"      Text="0.00"
TextColor="Navy"            Style="{StaticResource
baseLabelStyle}" Grid.Row="2" Grid.Column="1" />
```

٥- قم بتشغيل التطبيق، تحقق من أن قيم Tip and Total (التي تحتوي على القيم 0.00) لا تزال منسقة بشكل صحيح.

استخدام خاصية وراثة الأنماط Use style inheritance

أنت الآن جاهز لإعادة بناء التعليمات لأنماطك باستخدام الوراثة inheritance تتيح لك إعادة بناء التعليمات البرمجية التخلص من الاستخدام المتكرر لـ setter

١- افتح ملف StandardTipPage.xaml

٢- حدد موقع نمط infoLabelStyle style في قاموس موارد resource dictionary الصفحة، انقل هذا النمط أسفل baseLabelStyle في قاموس الموارد.

٣- تعيين الخاصية BasedOn لنمط infoLabelStyle إلى baseLabelStyle قم بإزالة setter لـ FontSize لم تعد بحاجة إليه لأن هذا النمط يرث الآن الإعداد FontSize من النمط الأساسي base style

```
<ContentPage.Resources>
 <ResourceDictionary>
        ...
   <Style x:Key="baseLabelStyle"
TargetType="Label">
   <Setter Property="FontSize"
Value="{StaticResource fontSize}" />
  </Style>
    <Style x:Key="infoLabelStyle"
BasedOn="{StaticResource baseLabelStyle}"
TargetType="Label">
 <Setter Property="FontAttributes" Value="Bold" />
  </Style>
 <ResourceDictionary>
</ContentPage.Resources>
```

ملاحظة:

ترتيب الموارد the resources في قاموس الموارد the resource dictionary أمرًا مهمًا، يجب تعريف baseLabelStyle style قبل أن تعمل أي أنماط أخرى تشير إليه، بخلاف ذلك لن يعمل توريث النمط style inheritance

٤- قم بتشغيل التطبيق وتحقق من أن أنماط styles الخاصة labels and calculated amounts لا تزال منسقة بشكل صحيح.

٨ إنشاء موارد على مستوى التطبيق application-wide resources واستخدامها

يمثل تحديد الموارد والأنماط على صفحة XAML وسيلة رائعة لتقليل معدل تكرار التعليمات البرمجية، ومع ذلك، توجد مشكلة، فتلك الموارد والأنماط لا تتوفر إلا على صفحة XAML هذه بالتحديد، ولا تكفي قواميس الموارد على مستوى الصفحة للسماح لك بتجنب تكرار التعليمات في أحد التطبيقات عندما تكون لديك عدة صفحات، في هذا الدرس، سترى كيفية مشاركة الموارد والأنماط عبر جميع الصفحات في تطبيقك.

الأماكن التي تتوفر فيها قواميس الموارد resource dictionaries

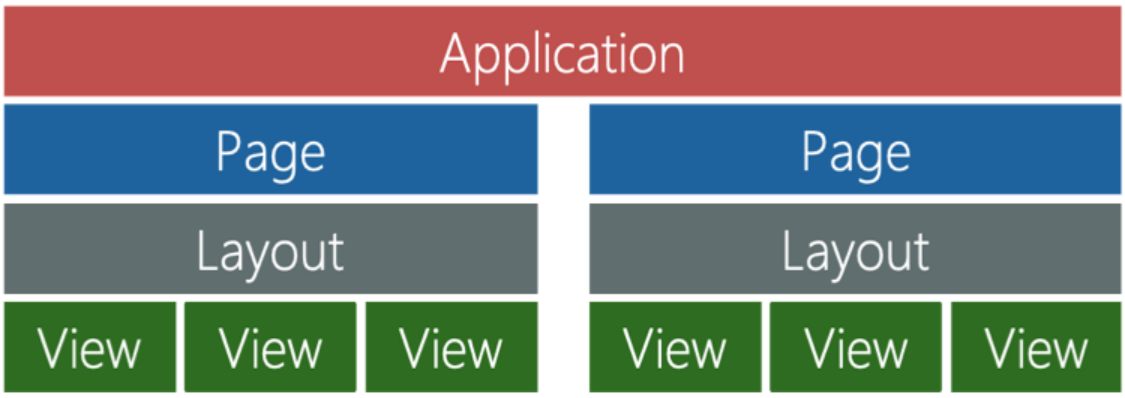
تحدد فئة The VisualElement class الخاصية Resources ترث عناصر التحكم والصفحات والتخطيطات Controls, pages, layouts من VisualElement لذا تحتوي جميعها على خاصية Resources التي يمكن أن تحتوي على قاموس مورد resource dictionary

تعرف فئة The Application class أيضاً خاصية Resources لا يرث التطبيق من VisualElement لذلك يتم تعريف الخاصية كجزء من هذه class

يوضح الرسم التوضيحي التالي بنية تطبيق نموذجي، يحتوي كل عنصر من العناصر المعروضة على خاصية Resources التي يمكن أن تحتوي على قاموس مورد.

ملاحظة:

يوضح هذا الرسم التخطيطي وصفًا مبسطًا لكيفية تنظيم العناصر في التطبيق، في هذا الرسم التخطيطي، يشير مصطلح **View** إلى عنصر تحكم فردي مثل Button or Label والتي لا تعمل كحاوية لأي عناصر تحكم تابعة child controls أيضاً، يشير المصطلح **Layout** إلى حاوية مسؤولة عن تنظيم تخطيط عناصر التحكم التابعة child controls الخاصة بها، يمكن تداخل التخطيط. على سبيل المثال، يمكن الاحتفاظ بعنصر تحكم Grid ضمن عنصر تحكم StackLayout



كيفية تحديد الموارد والأنماط على مستوى التطبيق application-level resources and styles

يمكنك تعريف الموارد والأنماط resources and styles على مستوى التطبيق في ملف XAML المقترن بفئة التطبيق Application class توضح التعليمات التالية كيفية تعريف مورد Color في قاموس مورد التطبيق the application resource dictionary

```
<Application.Resources>
    <Color x:Key="MyTextColor">Blue</Color>
</Application.Resources>
```

كيف يحدد NET MAUI. موقع مورد أو نمط resource or style

لنفترض أنك قمت بتطبيق مورد على أحد عناصر التحكم كما هو موضح في التعليمات التالية:

```
<Label TextColor="{StaticResource MyTextColor}" ... />
```

يحتاج NET MAUI. إلى تحديد تعريف هذا المورد resource حتى يتمكن من تطبيق القيمة the value يمكن أن يحتوي تطبيق واحد على العديد من القواميس dictionaries ما هي القواميس التي سيبحث عنها NET MAUI. وبأي ترتيب؟ للإجابة على هذه الأسئلة، من المفيد التفكير في مثيلات VisualElement instances على الصفحات على أنها تشكل بنية تشبه الشجرة، يقع التطبيق في الجذر the root مع انتشار الصفحات والتخطيطات وطرق العرض أسفله، غالبًا ما يُطلق على هذه البنية اسم الشجرة المرئية the visual tree يمكن أن يحتوي كل عنصر في الشجرة على قاموس dictionary خاص به يمكن أن يحتوي على موارد resources تتجه خوارزمية البحث عن الأنماط في NET MAUI. إلى أعلى الشجرة المرئية:

١- ابدأ البحث باستخدام القاموس في مثيل VisualElement الذي تم تطبيق المورد عليه، في المثال السابق، يبدأ البحث بنوع الملصق the Label type إذا لم يكن هناك قاموس موارد، أو إذا كان يحتوي على قاموس ولكن المورد غير موجود، فإن البحث ينتقل إلى التالي.

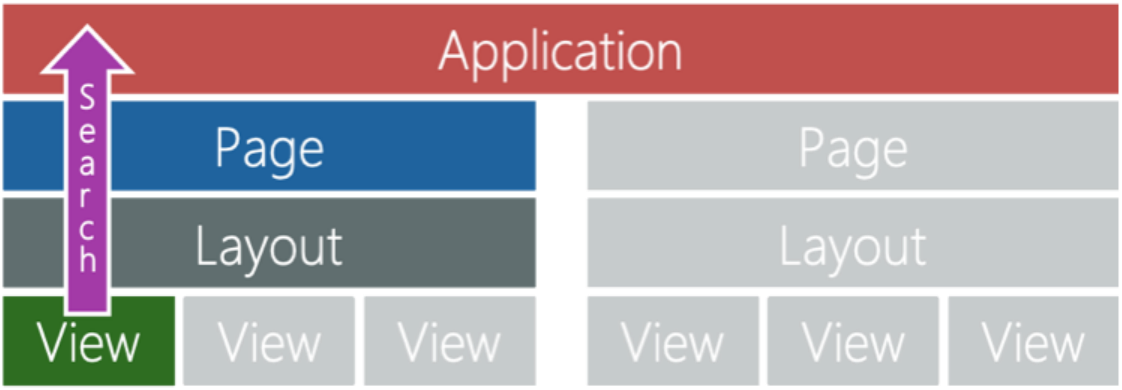
٢- انتقل إلى أصل عنصر التحكم the parent of the (control) وكرر عملية البحث، عادةً يكون المكان التالي الذي سيتم البحث فيه تخطيطاً layout

٣- تحقق من أصل التخطيط the parent of the layout عادةً ما يكون المكان التالي الذي سيتم البحث فيه هو الصفحة the page على الرغم من أنه إذا كان التخطيط

متداخلاً داخل تخطيط آخر (مثل Grid داخل StackLayout)، فسينقل البحث إلى أعلى الشجرة حتى يصل إلى التخطيط الأصل the parent layout

٤- ابحث في القاموس the dictionary عن فئة التطبيق the Application class

يقوم البحث بإرجاع العنصر الأول الذي تم العثور عليه بقيمة مفتاح مطابقة matching **x:Key** value تلخص الصورة التالية تسلسل البحث عن الموارد



في الممارسة العملية، يتجاهل معظم المطورين خاصية الموارد the Resources property في طرق العرض views والتخطيطات layouts فهم يستخدمون قواميس مستوى الصفحة the page-level dictionaries للأشياء التي يستخدمونها على صفحة واحدة، ويتم تعريف الموارد والأنماط Resources and styles التي يريدون مشاركتها عبر صفحات متعددة على مستوى التطبيق، ثم تحتاج عملية البحث إلى التحقق من قاموسين فقط: القاموس الموجود في مثيل الصفحة الحالية، والقاموس الموجود في التطبيق.

ملحوظة: إذا لم يتم العثور على مورد resource بالمفتاح المحدد the specified key فسيستخدم التطبيق القيم الافتراضية للتصميم default values for the styling

مفاتيح مكررة Duplicate keys

كل مثيل من ResourceDictionary مستقل، مما يعني أنه يمكن استخدام نفس قيمة x:Key في أكثر من قاموس. إن وجود نفس معرف x:Key في قواميس متعددة على مسار البحث لا يسبب خطأ، المورد المقترن بأول قيمة x:Key مطابقة على المسار هو المورد المستخدم.

على سبيل المثال، افترض أن لديك المورد التالي المحدد في فئة Application class

```
<Application.Resources>
    <x:String x:Key="msg">Two</x:String>
</Application.Resources>
```

ثم تقوم بتعريف المورد التالي في ContentPage وتطبيقه على Label على نفس الصفحة:

```
<ContentPage.Resources>
    <x:String x:Key="msg">One</x:String>
</ContentPage.Resources>
...
<Label Text="{StaticResource msg}">
```

نظرا لاستخدام أول قيمة مفتاح مطابقة matching x:Key value يتم تعيين الخاصية Text إلى One

٩ تمرين: استخدام الموارد على مستوى التطبيق application-wide resources

الهدف من هذا التمرين هو إتاحة الموارد عبر صفحات متعددة عن طريق نقلها إلى قاموس مورد resource dictionary في فئة تطبيق the Tip Calculator Application class

يُعد هذا التمرين استمراراً للتمرين السابق، إما أن تستخدم الحل الموجود كنقطة بداية لهذه الخطوات، أو افتح مشروع TipCalculator في المجلد exercise4/TipCalculator في المستودع الذي نسخته في التمرين الأول.

التحقق من موارد على مستوى الصفحة Verify page-level resources

فلنتأكد من أن الموارد المحددة في إحدى الصفحات غير متاحة في صفحة أخرى، في نهاية هذا القسم، لن يعمل تطبيقك بشكل صحيح، ومع ذلك، يمكنك إصلاحه في القسم التالي.

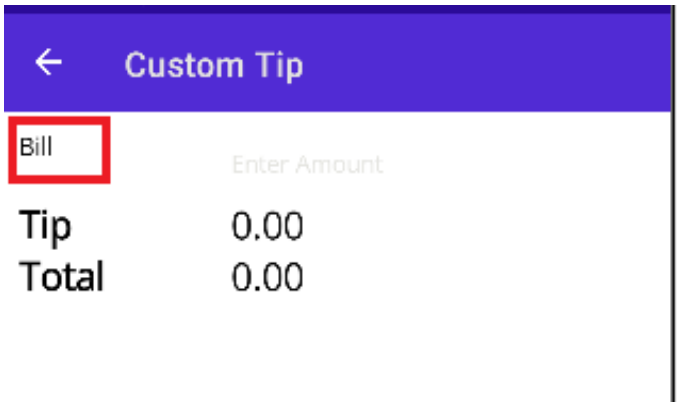
١- في مشروع TipCalculator افتح الملف CustomTipPage.xaml

٢- قم بتعيين مورد infoLabelStyle resource كنمط style لـ billLabel label وقم بإزالة الإعدادات الموجودة لخصائص FontSize and FontAttributes

```
<Label x:Name="billLabel" Text="Bill"
Style="{StaticResource infoLabelStyle}"
Grid.Row="0" Grid.Column="0" />
```

٣- شغل التطبيق.

حدد Use Custom Calculator انظر إلى CustomTipPage لعرض صفحة Bill label يجب أن يكون حجم الخط font size أصغر من labels الأخرى وليس غامقاً bold يرجع هذا السلوك إلى أن الصفحة لم تعثر على مورد يسمى infoLabelStyle (موجود في قاموس الموارد لصفحة مختلفة) لذلك يتم استخدام القيم الافتراضية لحجم الخط وسمات الخط font size and font attributes



إنشاء قاموس dictionary للموارد على مستوى التطبيق application-level resources

فلننشئ قاموس موارد على مستوى التطبيق للاحتفاظ بالموارد لاستخدامها على صفحات متعددة.

١- افتح ملف App.xaml لاحظ أن هذا الملف يحتوي حالياً على قاموس موارد مع بعض قواميس الموارد والأنماط الموجودة التي يتم استخدامها بشكل افتراضي لعناصر التحكم المضمنة في .NET MAUI. لمشاهدة كافة الأنماط المضمنة بشكل افتراضي، اعرض الملف Resources/Styles.xaml

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
   xmlns:local="clr-namespace:TipCalculator"
    x:Class="TipCalculator.App">
  <Application.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary
Source="Resources/Colors.xaml" />
        <ResourceDictionary
Source="Resources/Styles.xaml" />
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

٢- افتح ملف StandardTipPage.xaml وانقل مورد fontSize resource وأنماط baseLabelStyle and infoLabelStyle styles إلى قاموس الموارد resource dictionary في ملف App.xaml ضعها بعد الأنماط الموجودة بحيث يبدو ملف App.Xaml مثل المثال التالي:

```
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
                ...>
    <Application.Resources>
        <ResourceDictionary>
          <ResourceDictionary.MergedDictionaries>
```

```
                <ResourceDictionary
Source="Resources/Colors.xaml" />
                <ResourceDictionary
Source="Resources/Styles.xaml" />
            </ResourceDictionary.MergedDictionaries>

            <Color x:Key="bgColor">#C0C0C0</Color>
            <Color x:Key="fgColor">#0000AD</Color>
            <x:Double x:Key="fontSize">22</x:Double>

                <Style x:Key="baseLabelStyle"
TargetType="Label">
                <Setter Property="FontSize"
Value="{StaticResource fontSize}" />
                </Style>
                <Style x:Key="infoLabelStyle"
BasedOn="{StaticResource baseLabelStyle}"
TargetType="Label">
                <Setter Property="FontAttributes"
Value="Bold" />
                </Style>
            </ResourceDictionary>
        </Application.Resources>
</Application>
```

٣- شغل التطبيق.

٤- حدد علامة التبويب Use Custom Calculator tab وتحقق من أن Bill label قد تم تصميمه الآن بشكل صحيح.

١٠ الملخص

في هذه الوحدة، رأيت كيفية استخدام الموارد والأنماط resources and styles في تطبيقات .NET MAUI applications. وكان الهدف هو تبسيط تحديثات واجهة المستخدم لتطبيق TipCalculator كلما تغيرت العلامة التجارية للشركة، لقد استخدمت الميزات التالية لإعادة تنظيم التعليمات البرمجية بحيث تكون التحديثات إلى واجهة المستخدم سريعة ودقيقة:

- الموارد Resources قمت بإنشاء ثوابت رمزية symbolic constants لقيم الحجم واللون size and color values
- الأنماط Styles قمت بتعريف مظهر عنصر تحكم باستخدام مجموعة من قيم الخصائص group of property values

تتيح لك الموارد والأنماط تجميع كل التعريفات التي تؤثر على العلامة التجارية لواجهة المستخدم الخاصة بك، تسهل هذه الميزات تحقيق التناسق عبر جميع عناصر واجهة المستخدم، كما تتيح لك إجراء التحديثات بسرعة ودون أخطاء، وكمكافأة، أصبح من الأسهل قراءة XAML لصفحاتك لأنك تستطيع الرجوع إلى خاصية Style بدلاً من إعدادات الخصائص الفردية.

وبشكل أكثر تحديدًا، تعلمت في هذه الوحدة كيفية:

- إنشاء موارد ثابتة static resources واستخدامها في واجهة مستخدم .NET MAUI XAML user interface
- إنشاء الموارد التفاعلية dynamic resources واستخدامها.
- إنشاء واجهة مستخدم متناسقة باستخدام الأنماط styles
- إنشاء واستخدام الموارد على مستوى التطبيق application-wide resources

الوحدة الخامسة

إنشاء تطبيقات .NET MAUI. متعددة الصفحات باستخدام التنقل بين علامات التبويب والقوائم المنبثقة

استخدم .NET MAUI shell لإنشاء تطبيقات متعددة الصفحات multi-page باستخدام علامات التبويب tabs والقوائم المنبثقة flyout navigation

الأهداف التعليمية

في نهاية هذه الوحدة، ستتمكن من:

- تنفيذ التنقل عبر علامات التبويب tab navigation باستخدام .NET MAUI Shell
- التنقل بين الصفحات داخل الصفحات المبوبة tabbed pages
- تنفيذ التنقل المنبثق flyout navigation باستخدام .NET MAUI Shell.

محتويات الوحدة:

١- المقدمة

٢- تنفيذ التنقل المنبثق flyout navigation

٣- تمرين: تنفيذ التنقل المنبثق flyout navigation

٤- تنفيذ التنقل بين علامات التبويب tab navigation باستخدام .NET MAUI Shell

٥- تمرين: تنفيذ التنقل بين علامات التبويب tab navigation

٦- استخدام الصفحات المبوبة tabbed pages مع الصفحات الموجودة في مجموعة التنقل the navigation stack

٧- تمرين: استخدام الصفحات المبوبة tabbed pages مع صفحات التنقل navigation pages

٨- الملخص

١ المقدمة

يعد تحديد كيفية تنقل المستخدم بين الصفحات جزءًا من تخطيط بنية التطبيق، هل يجب على المستخدمين الانتقال إلى الأمام والخلف عبر سلسلة من الصفحات؟ هل يحتوي تطبيقك على صفحة بدء واحدة، أم أن هناك العديد من صفحات على مستوى عالٍ بنفس القدر من الأهمية؟ يجب أن يتوافق اختيارك للتنقل بشكل جيد مع محتوى تطبيقك، ويجب أن تشعر أيضًا أنه أصلي ومتوافق مع كل نظام تشغيل تستهدفه.

لنفترض أنك تكتب تطبيقاً لمتحف فلكي باستخدام واجهة مستخدم التطبيق متعددة الأنظمة .NET Multi-platform App UI (MAUI) وتريد إثارة الاهتمام بعلم الفلك من خلال تقديم الحقائق والمفاهيم الفلكية ذات الصلة بالحياة اليومية، يجب أن يحتوي التطبيق على صفحات تغطي شروق الشمس وغروبها، والمراحل القمرية، والأجرام الفلكية، وصفحة حول التطبيق. تتمثل مهمتك في تصميم وتنفيذ نمط تنقل بديهي لهذه الصفحات.

في هذه الوحدة، ستتعلم كيفية استخدام الصفحات المبوبة tabbed pages لتنفيذ تطبيق يعرض البيانات على صفحات متعددة، يمكنك معرفة متى يكون نهج التنقل هذا مناسباً، وكيف يمكنك تخصيص سلوك الصفحات المبوبة، يمكنك أيضاً التعرف على كيفية دمج الصفحات المبوبة مع صفحات التنقل navigation pages التي تستخدم مكدس التنقل the navigation stack

٢ تنفيذ التنقل المنبثق flyout navigation

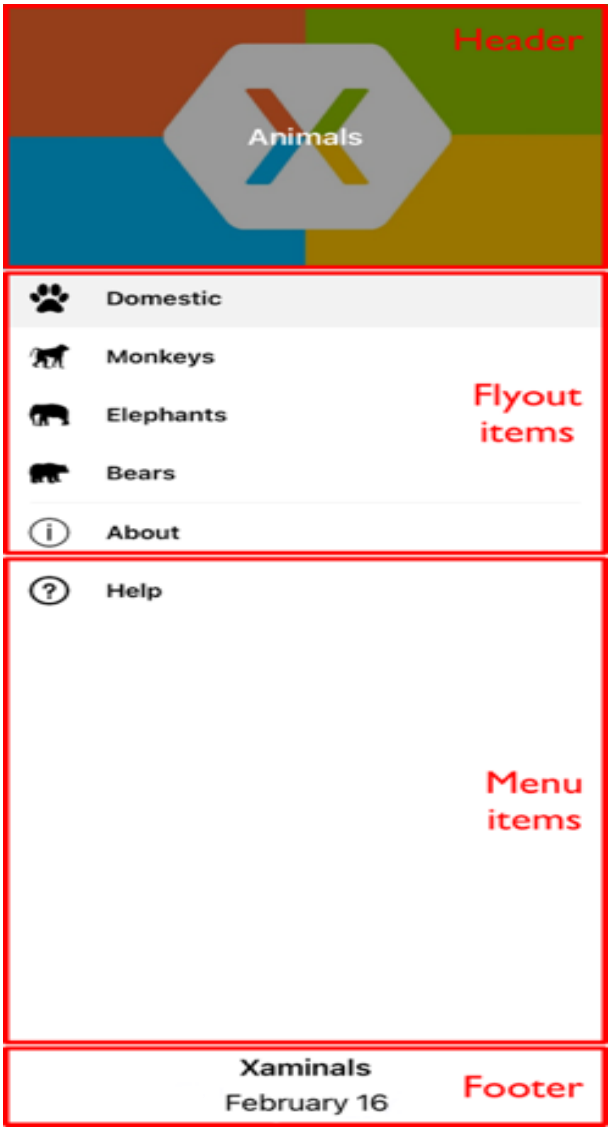
التنقل المنبثق Flyout navigation هو نوع من التنقل حيث تنزلق نافذة عناصر القائمة (أو تطير للخارج) من جانب شاشة الجهاز، يتم استدعاؤها من خلال النقر على قائمة menu "hamburger" أيقونة تحتوي على ثلاثة خطوط أفقية متراصة فوق بعضها البعض.

في هذا الدرس، ستتعلم كيفية إنشاء تطبيق يطبق التنقل المنبثق flyout navigation في واجهة مستخدم تطبيق متعددة الأنظمة .NET (MAUI).

ما هو التنقل المنبثق flyout navigation

يعرض التنقل المنبثق Flyout navigation قائمة توفر وسيلة سريعة لتبديل السياق داخل التطبيق الخاص بك. تتكون القائمة المنبثقة The flyout menu من عدة أجزاء Header, FlyoutItems, MenuItems, Footer

تعرض الصورة التالية مثالاً مرئياً لأجزاء القائمة المنبثقة:



نظراً لأن القائمة المنبثقة غير مرئية دائماً، يمكن استخدامها لتبديل السياق بين أجزاء مختلفة مفاهيميًا من تطبيقك، على سبيل المثال، يمكن أن يؤدي عنصر قائمة منبثقة إلى صفحة إدخال بيانات (أو صفحات) وآخر إلى صفحة حول التطبيق.

التنقل المنبثق Flyout navigation in a .NET MAUI app

تستخدم فئة the `FlyoutItem` class لتنفيذ التنقل المنبثق في `flyout navigation in .NET MAUI` ال `FlyoutItem` هو جزء من نموذج تطوير تطبيقات `Shell app development` الذي توفره `.NET MAUI`.

يحدث التنقل باستخدام نافذة منبثقة في `.NET MAUI` عند النقر فوق عنصر FlyoutItem يقوم عنصر FlyoutItem تلقائيًا بتبديل المحتوى المعروض في تطبيقك، يمكنك تحديد المحتوى الذي يتم عرضه عند النقر فوق عنصر FlyoutItem عن طريق تعيين خاصية `property` `ShellContent` الخاصة به، تشير هذه الخاصية إلى صفحة في تطبيقك.

يجب استضافة FlyoutItem في صفحة Shell والتي تعمل كصفحة رئيسية لتطبيقك. ويمكنك الحصول على عدد لا حصر له من FlyoutItems كما تريد.

ينشئ المثال التالي قائمة منبثقة تحتوي على عنصرين منبثقة:

```
<Shell xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:controls="clr-namespace:Xaminals.Controls"
   xmlns:views="clr-namespace:Xaminals.Views"
   x:Class="Xaminals.AppShell">
 <FlyoutItem Title="Cats"
                 Icon="cat.png">
    <Tab>
      <ShellContent ContentTemplate="{DataTemplate
views:CatsPage}" />
    </Tab>
  </FlyoutItem>
  <FlyoutItem Title="Dogs"
                 Icon="dog.png">
    <Tab>
      <ShellContent ContentTemplate="{DataTemplate
views:DogsPage}" />
    </Tab>
```

```
 </FlyoutItem>
</Shell>
```

إنشاء قائمة منبثقة Create a flyout

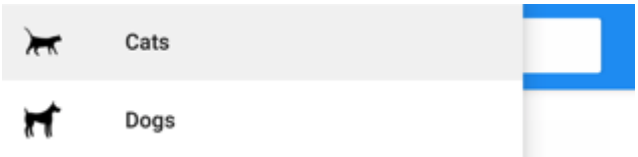
يمكن إضافة عنصر واحد أو أكثر من العناصر المنبثقة flyout items إلى القائمة المنبثقة the flyout يمثل الكائن `FlyoutItem object` كل عنصر منبثق، يجب أن يكون كل كائن `FlyoutItem object` فرعيًا a child لكائن `Shell object` الفرعي الذي يعمل كصفحة رئيسية `MainPage` لتطبيقك.

يحتوي `Shell object` على عوامل تحويل ضمنية، تمكن من تبسيط التسلسل الهرمي المرئي لـ Shell يصبح هذا التبسيط ممكنًا لأن `Shell object` الفرعي لا يمكنه أبدًا أن يحتوي إلا على `FlyoutItem objects` أو `TabBar object` والذي يمكن أن يحتوي فقط على `Tab objects` والتي لا يمكنها أبدًا أن تحتوي إلا على `ShellContent objects`

يمكن استخدام عوامل التحويل الضمنية implicit conversion operators هذه لإزالة FlyoutItem and Tab objects من المثال السابق:

```
<Shell xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:controls="clr-namespace:Xaminals.Controls"
  xmlns:views="clr-namespace:Xaminals.Views"
  x:Class="Xaminals.AppShell">
 <ShellContent Title="Cats"
   Icon="cat.png"
   ContentTemplate="{DataTemplate views:CatsPage}" />
 <ShellContent Title="Dogs"
   Icon="dog.png"
   ContentTemplate="{DataTemplate views:DogsPage}" />
</Shell>
```

ينتج عن هذه التعليمة البرمجية ظهور قائمة منبثقة تحتوي على عنصرين، يتم عرض `CatsPage` افتراضيًا عند فتح التطبيق، يؤدي النقر على العنصر الثاني إلى عرض `DogsPage`



عناصر القائمة المنبثقة Flyout menu items

يمكن إضافة عناصر القائمة Menu items بشكل اختياري إلى القائمة المنبثقة the flyout يمثل object `MenuItem` كل عنصر من عناصر القائمة، تشبه عناصر القائمة الأزرار، حيث يؤدي النقر فوق أحدها إلى حدوث إجراء، بدلاً من عرض صفحة.

يعتمد موضع objects `MenuItem` في القائمة المنبثقة على ترتيب إعلانها في التسلسل الهرمي المرئي لـ Shell لذلك، أي objects `MenuItem` معلنة قبل objects `FlyoutItem` ستظهر قبل objects `FlyoutItem` الموجودة في القائمة المنبثقة.

وأي objects `MenuItem` معلنة بعد objects `FlyoutItem` ستظهر بعد objects `FlyoutItem` الموجودة في القائمة المنبثقة.

```
<Shell ...>
    ...
    <MenuItem Text="Help"
      IconImageSource="help.png"
      Command="{Binding HelpCommand}"

CommandParameter="https://learn.microsoft.com/dotn
et/maui/fundamentals/shell" />
</Shell>
```

رأس وتذييل القائمة المنبثقة Flyout header and footer

يُعد رأس الصفحة المنبثق The flyout header هو المحتوى الذي يظهر اختياريًا في أعلى القائمة المنبثقة the top of the flyout يمكنك تحديد مظهر الرأس the appearance of the header عن طريق تعيين كائن باستخدام الخاصية القابلة للربط bindable property `Shell.FlyoutHeader`

```
<Shell ...>
    <Shell.FlyoutHeader>
        <Grid>
            <Image Source="header-image.png" />
        </Grid>
    </Shell.FlyoutHeader>
</Shell>
```

يُعد تذييل القائمة المنبثقة The flyout footer هو المحتوى الذي يظهر اختياريًا في أسفل القائمة المنبثقة the bottom of the flyout يمكنك تحديد مظهر التذييل the appearance of the footer عن طريق تعيين كائن باستخدام الخاصية القابلة للربط `Shell.FlyoutFooter` bindable property

```
<Shell ...>
    <Shell.FlyoutFooter>
        <Grid>
            <Image Source="footer-image.png" />
        </Grid>
    </Shell.FlyoutFooter>
</Shell>
```

٣ تمرين: تنفيذ التنقل المنبثق flyout navigation

في السيناريو النموذجي، لديك تطبيق .NET MAUI. يحتوي على صفحات لعرض معلومات حول الأجرام الفلكية، ومراحل القمر، وأوقات شروق الشمس وغروبها. يتضمن التطبيق أيضًا صفحة "حول" حاليًا، كل هذه الصفحات مستقلة، ولكنك تريد توفير طريقة منطقية للمستخدم للتنقل بينها.

في هذا التمرين، يمكنك إضافة التنقل المنبثق flyout navigation إلى التطبيق.

تستخدم هذه الوحدة حزمة .NET 9.0 SDK تأكد من تثبيت .NET 9.0 من خلال تشغيل الأمر التالي في موجة الأوامر لديك:

```
dotnet --list-sdks
```

تأكد أن الإخراج مشابه للتالي:

```
8.0.100 [C:\Program Files\dotnet\sdk]
9.0.100 [C:\Program Files\dotnet\sdk]
```

تأكد من إدراج إصدار يبدأ بالرقم 9 إذا لم يتم إدراج أي إصدار أو لم يتم العثور على الأمر، فقم بتثبيت أحدث مجموعة أدوات تطوير install the most recent .NET 9.0 SDK

افتح the starter solution

١- نسخ أو تحميل exercise repo

٢- انتقل إلى مجلد exercise1 في المستودع المستنسخ، ثم انتقل إلى مجلد start

٣- استخدم Visual Studio لفتح Astronomy.sln solution أو المجلد في Visual Studio Code

٤- في نافذة مستكشف الحلول Solution Explorer في Astronomy project قم بتوسيع مجلد Pages يحتوي هذا المجلد على الصفحات التالية:

- حول الصفحة AboutPage تعرض هذه الصفحة معلومات التطبيق.
- MoonPhasePage تعرض هذه الصفحة معلومات محددة حول مراحل القمر كما نراها من الأرض.
- SunrisePage تعرض هذه الصفحة أوقات شروق وغروب الشمس للمواقع على الأرض.

يتم توفير البيانات بواسطة خدمة ويب Sunrise Sunset web service

٥- قم ببناء التطبيق وتشغيله. عند بدء تشغيل التطبيق، يتم عرض MoonPhasePage ولكن لا توجد حاليًا أي وسيلة لتمكين المستخدم من الانتقال إلى الصفحات الأخرى.

تظهر الصورة التالية التطبيق يعمل على محاكي Android

٦- أغلق التطبيق وارجع إلى Visual Studio أو Visual Studio Code

إضافة التنقل المنبثق

١- في نافذة Solutions Explorer افتح صفحة AppShell.xaml

٢- في محرر علامات XAML markup editor قم بإحاطة العنصر الموجود `<ShellContent>` بعنصر `<FlyoutItem>` اضبط الخاصية `Title` property للعنصر `<Flyout>` ليكون Moon Phase يجب أن تبدو العلامات كما يلي:

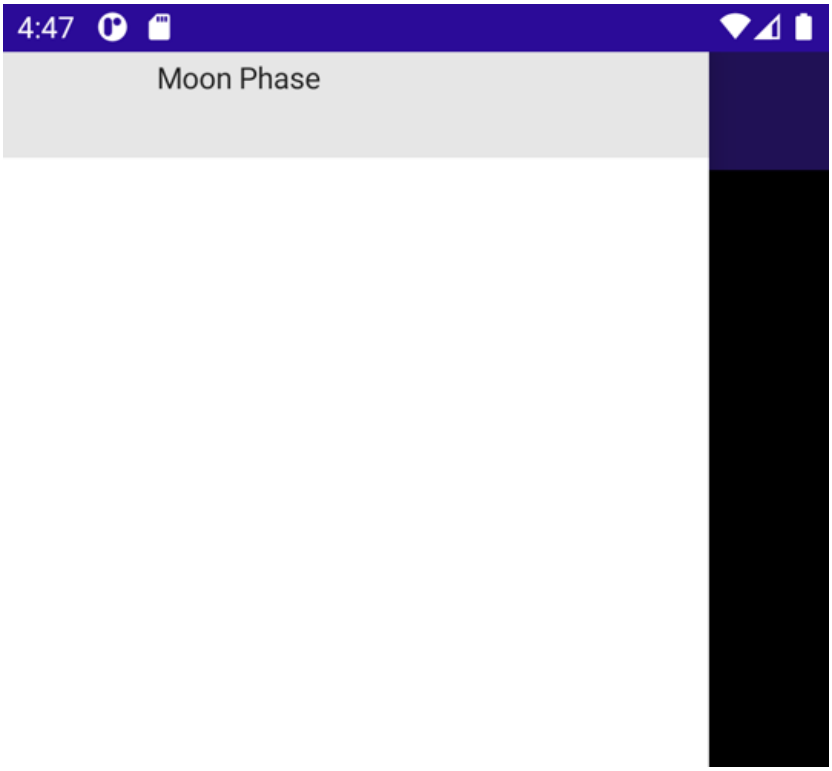
```
<FlyoutItem Title="Moon Phase">
 <ShellContent
 ContentTemplate="{DataTemplate local:MoonPhasePage}"/>
</FlyoutItem>
```

٣- أضف خاصية property FlyoutIcon إلى عقدة node <Shell> لعرض صورة. بشكل افتراضي، تعرض هذه الخاصية ثلاثة أشرطة أفقية horizontal bars ولكن يمكننا تغييرها إلى أي شيء نريده، يجب أن تبدو العلامات على النحو التالي:

```
<Shell
  x:Class="Astronomy.AppShell"
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:Astronomy.Pages"
   FlyoutIcon="moon.png">
```

٤- قم بتشغيل التطبيق: يجب أن تشاهد الآن صورة القمر في الزاوية اليسرى العليا the upper left corner من التطبيق.

اضغط على الأيقونة وتظهر القائمة المنبثقة the flyout

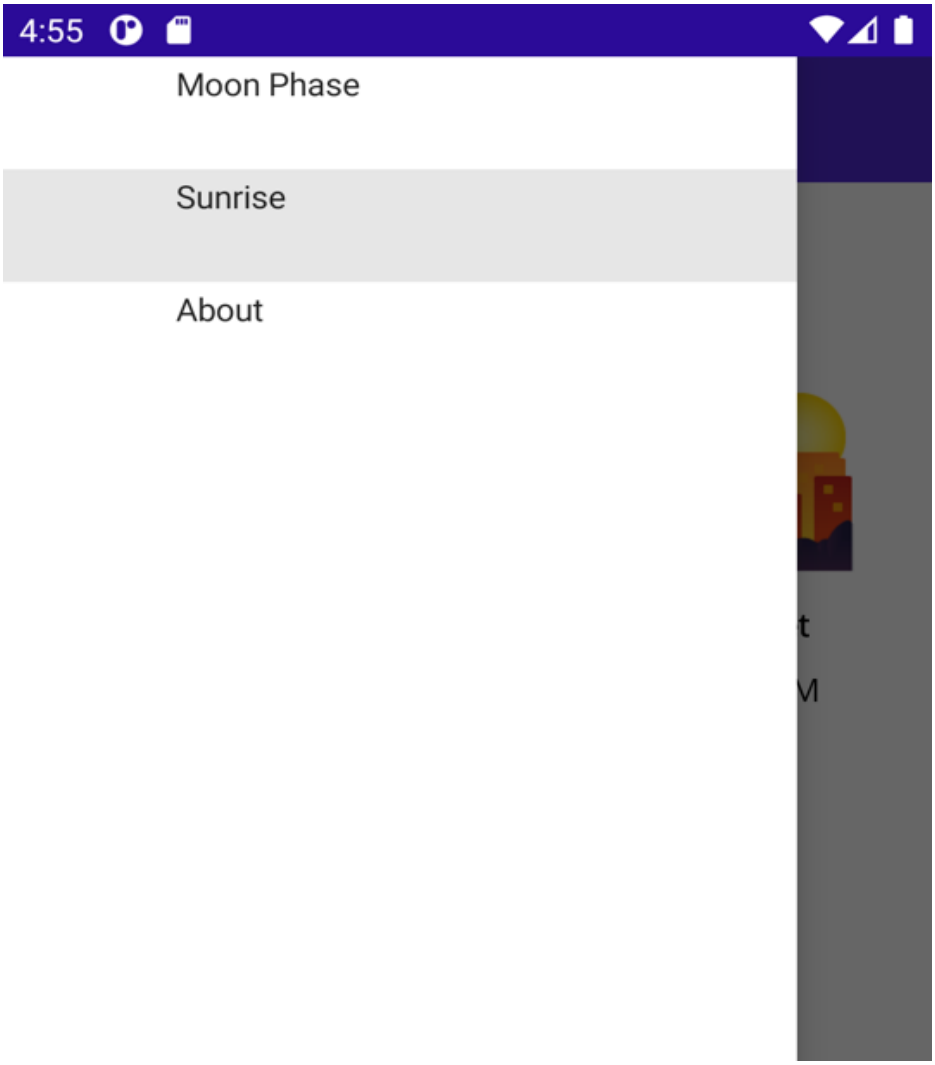


٥- الآن أضف المزيد من خيارات القائمة المنبثقة. أنشئ `<FlyoutItem>` جديدًا، أسفل العنصر الذي أنشأته للتو واضبط عنوانه `Title` على Sunrise يجب أن يشير `SunrisePage` page الخاص به إلى صفحة `ShellContent`

٦- أضف `<FlyoutItem>` آخر، واضبط عنوانه `Title` على About هذه المرة قم بتعيين `ShellContent` إلى `AboutPage` يجب أن يبدو XAML لهذين العنصرين كما يلي:

```
<FlyoutItem Title="Sunrise">
  <ShellContent
  ContentTemplate="{DataTemplate local:SunrisePage}"/>
</FlyoutItem>

<FlyoutItem Title="About">
  <ShellContent
  ContentTemplate="{DataTemplate local:AboutPage}"/>
</FlyoutItem>
```

٧- قم بتشغيل التطبيق مرة أخرى، وستجد الآن ثلاث خيارات في القائمة المنبثقة. يؤدي النقر على عنصر القائمة المنبثقة إلى عرض الصفحة الخاصة به.



ملاحظة:

إذا كنت تعمل على نظام تشغيل آخر غير Windows فقد تحتاج إلى تمكين إذن الموقع لتطبيقك، على هذا النظام لكي تعمل صفحة Sunrise/Sunset Times على سبيل المثال، على جهاز Android قم بتعيين الوصول إلى الموقع على "السماح" فقط أثناء استخدام التطبيق.

إضافة أيقونات icons

ربما لاحظت أن عناصر القائمة المنبثقة تبدو فارغة بعض الشيء. يمكنك إضافة أيقونات إلى عناصر القائمة المنبثقة the flyout items باستخدام الخاصية Icon

تمت إضافة بعض الصور بالفعل إلى مجلد Resources\Images لاستخدامها.

١- تعيين الخاصية `Icon property` الأولى `FlyoutItem` إلى moon.png

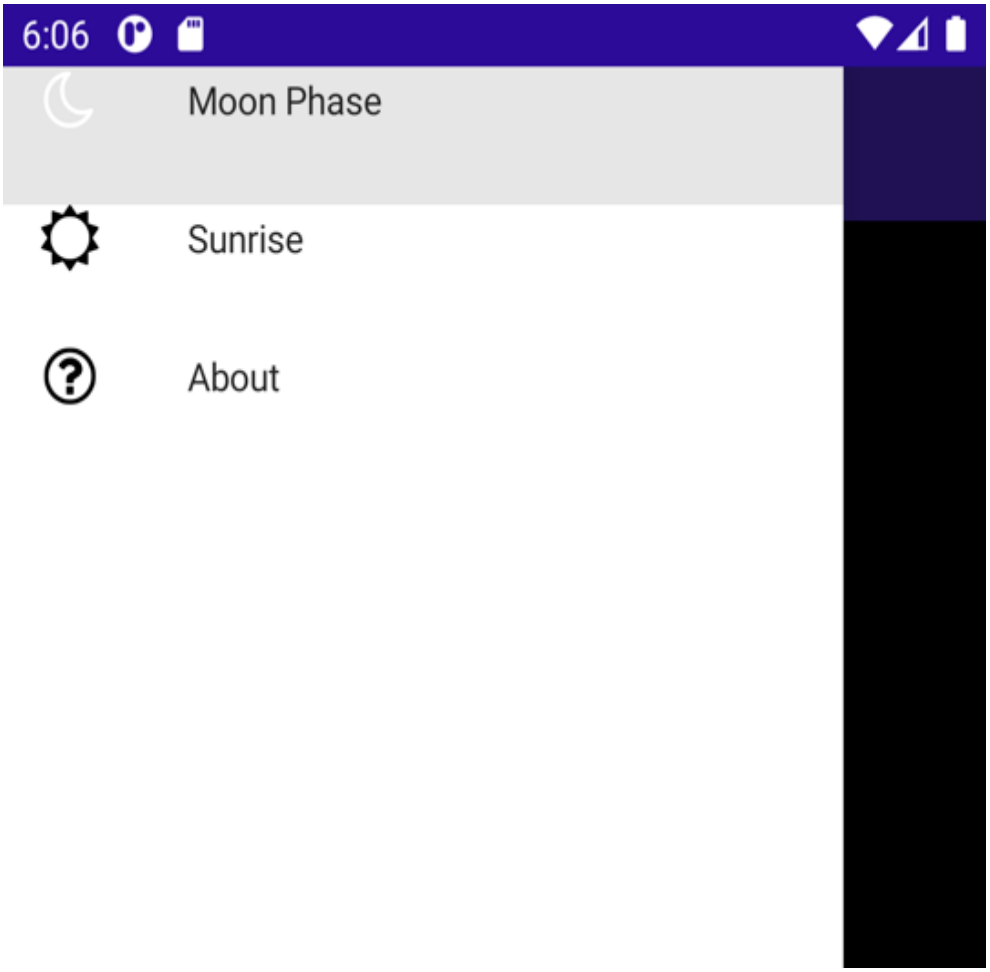
```
<FlyoutItem Title="Moon Phase" Icon="moon.png">
  <ShellContent
```

```
 ContentTemplate="{DataTemplate local:MoonPhasePage}"
/>
</FlyoutItem>
```

٢- كرر ذلك مع العناصر المنبثقة الأخرى، باستخدام sun.png and question.png على التوالي.

```
<FlyoutItem Title="Sunrise" Icon="sun.png">
  <ShellContent
 ContentTemplate="{DataTemplate local:SunrisePage}"/>
</FlyoutItem>

<FlyoutItem Title="About" Icon="question.png">
  <ShellContent
 ContentTemplate="{DataTemplate local:AboutPage}"/>
</FlyoutItem>
```

٣- شغل التطبيق وافتح القائمة المنبثقة open the flyout يحتوي كل عنصر قائمة منبثقة الآن على أيقونة مقترنة به.



إضافة رأس قائمة منبثقة flyout header

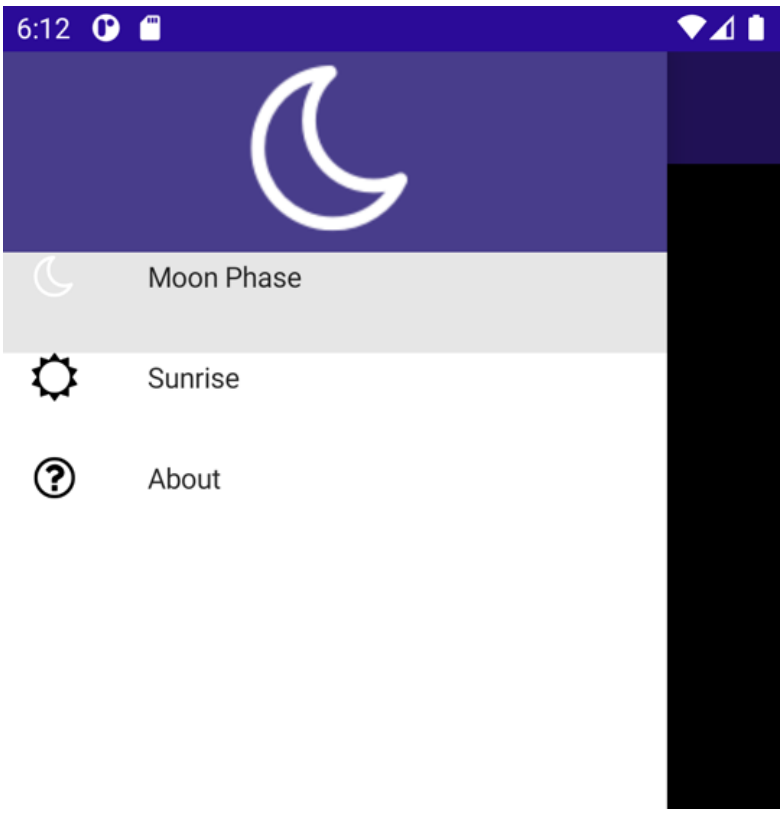
توجد عناصر القائمة المنبثقة The flyout items في أعلى القائمة المنبثقة the top of the flyout menu مما يجعل التمييز بينها أمرًا صعبًا، يمكننا إضافة بعض المساحة إلى الأعلى، وحتى عرض كامل entire View باستخدام <Shell.FlyoutHeader>

١- إضافة رأس قائمة منبثقة flyout header كعنوان تابع للعقدة child of the <Shell> node

```
<Shell.FlyoutHeader>
</Shell.FlyoutHeader>
```

٢- يمكنك إنشاء أي تسلسل هرمي للعرض، ترغب فيه داخل <Shell.FlyoutHeader> دعونا نضع في Grid مع Image

```
<Shell.FlyoutHeader>
 <Grid HeightRequest="100"
BackgroundColor="DarkSlateBlue">
   <Image Source="moon.png" />
 </Grid>
</Shell.FlyoutHeader>
```



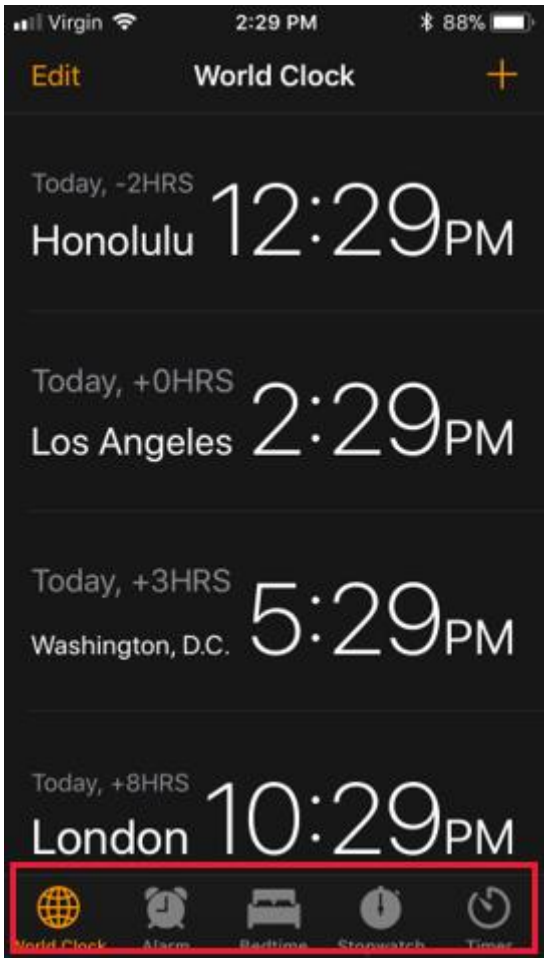
٤ تنفيذ التنقل بين علامات التبويب tab navigation باستخدام .NET MAUI Shell

التنقل بين علامات التبويب Tab navigation هو نمط تنقل حيث يتم عرض شريط علامات التبويب (صف من عناصر التحكم التي يمكن لمسها) بشكل دائم في أعلى الشاشة أو أسفلها the top or bottom of the screen يوفر التنقل بين علامات التبويب آلية للاختيار بين الصفحات في تطبيق متعدد الصفحات.

في هذا الدرس، ستتعلم كيفية إنشاء تطبيق ينفذ التنقل بين علامات التبويب implements tab navigation

ما المقصود بالتنقل بين علامات التبويب tab navigation

في التطبيق الذي يستخدم التنقل عبر علامات التبويب، تمثل كل علامة تبويب قسمًا أو صفحة معينة من التطبيق، يختار المستخدمون علامات التبويب داخل شريط علامات التبويب، للتنقل بين المحتوى في التطبيق، على سبيل المثال، يوضح الرسم التوضيحي التالي استخدام التنقل عبر علامات التبويب في تطبيق iOS Clock تتيح لك الايقونات المميزة أسفل الصفحة، التبديل بين طرق عرض views مختلفة، تتوافق هذه الايقونات مع علامات التبويب، وتكون طرق عرض views عبارة عن صفحات مبوبة:



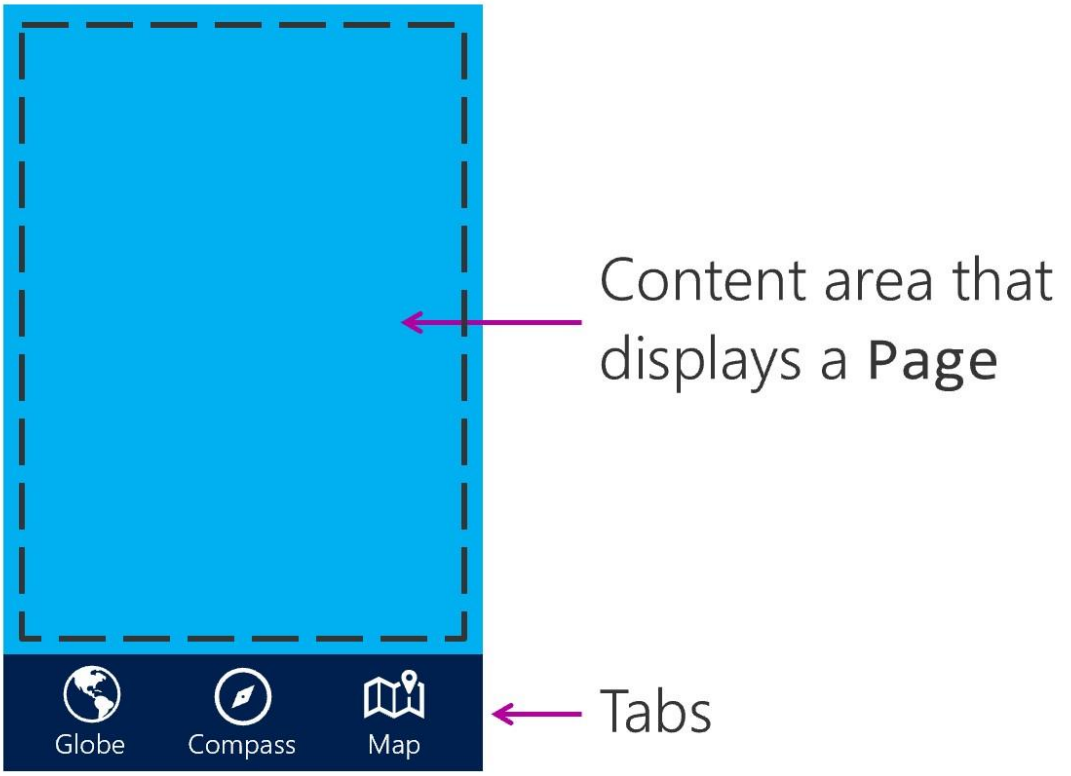
نظرًا لأن شريط علامات التبويب مرئي دائمًا، فإن التنقل بين علامات التبويب يسمح للمستخدمين بالتبديل بسرعة بين المحتوى في التطبيق. يعد التنقل بين علامات التبويب مثاليًا عندما يحتوي التطبيق على عدة أقسام من المحتمل أن يستخدمها المستخدم بشكل متكرر. تطبيقات الساعة هي مثال ممتاز على ذلك. من المرجح أن يتم استخدام أقسام الساعة والمنبه وساعة الإيقاف بشكل متكرر.

على الأجهزة المحمولة، عادةً ما يكون لشريط علامات التبويب مساحة محدودة، ويمكنه عرض عدد محدود فقط من علامات التبويب، اعتمادًا على حجم الجهاز واتجاهه، التوصية هي استخدام ثلاث إلى أربع علامات تبويب فقط، إذا قمت بتضمين المزيد من علامات التبويب، فلن يكون لديك مساحة كافية لعرض جميع علامات التبويب على جميع الأجهزة. تسمح أنظمة التشغيل التي يدعمها NET MAUI. بمساحة إضافية، توفر هذه المنطقة مساحة أكبر للوصول إلى علامات التبويب التي لا تتناسب مع الشاشة، ومع ذلك، يتطلب الانتقال إلى علامات التبويب الإضافية هذه خطوات إضافية من قبل المستخدم. هذه الأقسام أقل قابلية للاكتشاف.

إذا كنت بحاجة إلى أكثر من أربع علامات تبويب، ففكر في استخدام نمط تنقل آخر، مثل التنقل بين العناصر المنبثقة flyout item navigation بالإضافة إلى ذلك، لا يعد التنقل بين علامات التبويب tab navigation الخيار الأفضل إذا كانت بياناتك تشكل تسلسلًا هرميًا طبيعيًا من التفاصيل الرئيسية، في هذه الحالات، يجب أن تفكر في التنقل المجمع stack navigation

التنقل بين علامات التبويب Tab navigation في تطبيق NET MAUI.

يمكنك استخدام `TabBar` object لتنفيذ التنقل بين علامات التبويب في تطبيق `NET MAUI shell`. يعرض `TabBar` object مجموعة من علامات التبويب، ويبدل المحتوى المعروض تلقائياً عندما يحدد المستخدم علامة تبويب. يوضح الرسم التوضيحي التالي مناطق واجهة المستخدم.



لاستخدام علامات التبويب في تطبيق .NET MAUI shell قم بإنشاء مثيل للفئة TabBar class كفئة فرعية للفئة Shell class ثم أضف Tab child of the objects إلى داخل TabBar

يجب تعيين ShellContent object إلى ContentPage object

إنشاء TabbedPage

يمكنك إنشاء مثيل TabBar instance كعنصر فرعي Shell class أضف Tab objects كعناصر فرعية إلى TabBar حسب الحاجة. داخل Tab object يجب تعيين ShellContent object إلى ContentPage object

```
<Shell xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:views="clr-namespace:Xaminals.Views"
  x:Class="Xaminals.AppShell">
  <TabBar>
  <Tab Title="Moon Phase"
       Icon="moon.png">
       <ShellContent ContentTemplate="{DataTemplate
local:MoonPhasePage}" />
```

```
  </Tab>
  <Tab Title="Sunrise"
       Icon="sun.png">
       <ShellContent ContentTemplate="{DataTemplate
local:SunrisePage}" />
  </Tab>
  </TabBar>
</Shell>
```

علامات التبويب ضمن قائمة منبثقة Tabs within a flyout menu

يمكن لعنصر القائمة المنبثقة/المنسدلة flyout item فتح صفحة تحتوي على شريط علامات تبويب tab bar يعرض علامة تبويب واحدة أو أكثر.

لتنفيذ هذا التصميم، أضف `<ShellContent>` داخل `<FlyoutItem>` لكل علامة تبويب tab تريد عرضها.

قم بتعيين `Title` and `Icon` على `<ShellContent>` للتحكم في عنوان علامة التبويب وأيقونتها.

```
<FlyoutItem Title="Astronomy" Icon="moon.png">
 <ShellContent Title="Moon Phase" Icon="moon.png"
 ContentTemplate="{DataTemplate local:MoonPhasePage}"/>

 <ShellContent Title="Sunrise" Icon="sun.png"
 ContentTemplate="{DataTemplate local:SunrisePage}"/>
</FlyoutItem>

<FlyoutItem Title="About" Icon="question.png">
 <ShellContent
 ContentTemplate="{DataTemplate local:AboutPage}"/>
</FlyoutItem>
```

اختبار بسيط:

١- متى يكون التنقل عبر علامات التبويب tab navigation خيارًا جيدًا للتطبيق؟

حل الاختبار:

١- لديك عدد قليل من الصفحات في التطبيق، وكل الصفحات لها نفس الأهمية. يعمل التنقل بين علامات التبويب بشكل جيد في هذا السيناريو.

٥ تمرين: تنفيذ التنقل بين علامات التبويب tab navigation

في تطبيق علم الفلك the astronomy يُطلب منك دمج علامات التبويب مع القائمة المنبثقة للمساعدة في التنقل بين الصفحات المختلفة.

أول شيء تقرر القيام به هو إزالة جميع الصفحات من القائمة المنبثقة وإضافتها إلى TabBar حتى تتمكن من رؤية شكل التطبيق.

إضافة شريط تبويب TabBar

١- في نافذة Solution Explorer افتح صفحة page `AppShell.xaml`

٢- في صفحة XAML markup page احذف كل شيء داخل `<Shell>`

٣- إنشاء `<TabBar>` وإنشاء `<Tab>` empty فارغ.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
 x:Class="Astronomy.AppShell"
 xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
 xmlns:local="clr-namespace:Astronomy.Pages"
 FlyoutIcon="moon.png">

  <TabBar>
      <Tab>

      </Tab>
  </TabBar>
</Shell>
```

٤- بعد ذلك، أضف `ShellContent` إلى `Tab` وقم بتعيين محتواه إلى `MoonPhasePage`

```
<TabBar>
  <Tab>
   <ShellContent ContentTemplate="{DataTemplate
local:MoonPhasePage}" />
  </Tab>
</TabBar>
```

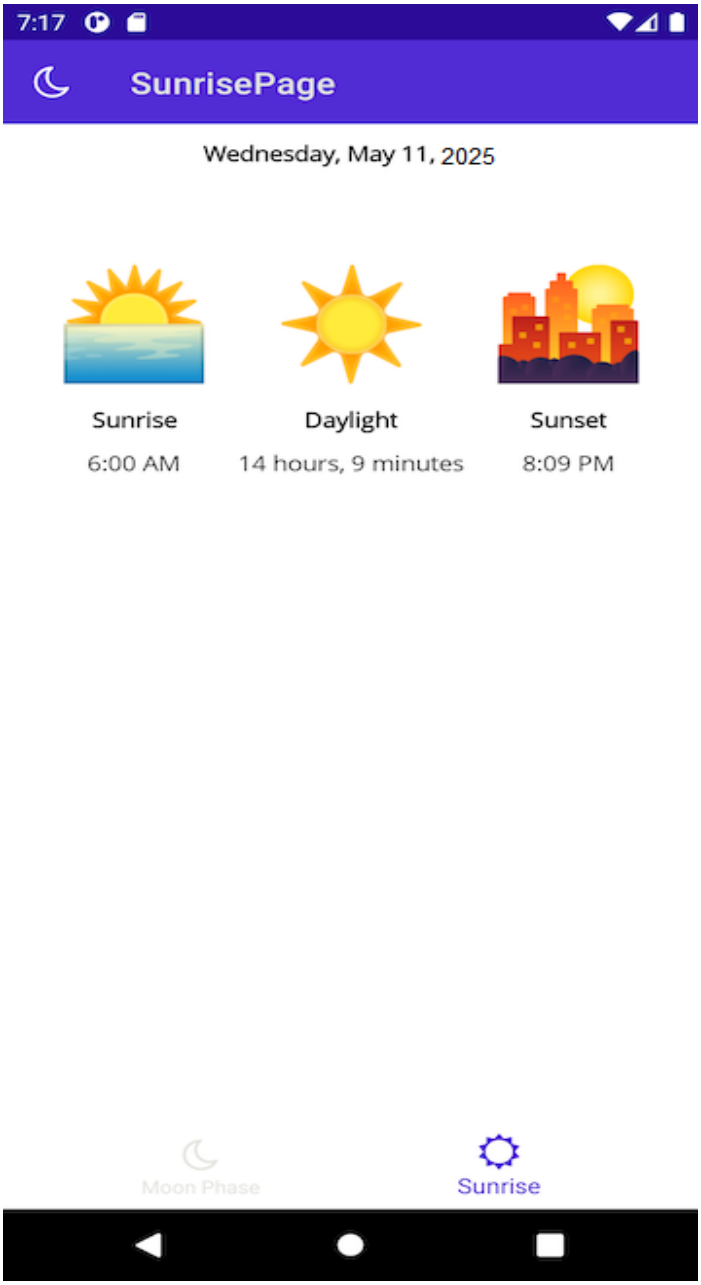
٥- الآن امنح علامة التبويب عنواناً ليتم عرضه، وأيقونة، باستخدام الخاصيتين `Title` and `Icon` properties

```
<Tab Title="Moon Phase" Icon="moon.png">
```

٦- أضف `Tab` أخرى لصفحة `SunrisePage` اضبط `Title` على sunrise وأيقونتها `Icon` على sun.png يبدو ملف XAML النهائي على النحو التالي:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
 x:Class="Astronomy.AppShell"
 xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
 xmlns:local="clr-namespace:Astronomy.Pages"
 FlyoutIcon="moon.png">

 <TabBar>
   <Tab Title="Moon Phase" Icon="moon.png">
      <ShellContent ContentTemplate="{DataTemplate
local:MoonPhasePage}" />
   </Tab>
   <Tab Title="Sunrise" Icon="sun.png">
       <ShellContent
ContentTemplate="{DataTemplate
local:SunrisePage}"/>
   </Tab>
 </TabBar>
</Shell>
```

٧- قم بتشغيل التطبيق لمعرفة كيف يبدو.



دمج صفحات علامة التبويب tab pages مع قائمة منبثقة flyout

قررت أنه من المنطقي أن يكون لديك صفحات طور القمر وشروق الشمس في نفس صفحة علامات التبويب tab page من المنطقي أيضاً الاحتفاظ بالصفحة حول the about page منفصلة. لذا قررت إضافة القائمة المنبثقة flyout مرة أخرى، يعرض عنصر القائمة المنبثقة الأول صفحة علامة التبويب the tab page والثاني صفحة حول the about page

١- احذف `TabBar` وكل العناصر التابعة the child items الموجودة فيه.

٢- في مكانه، أضف في `<FlyoutItem>` اضبط خاصية `Title` الخاصة به على Astronomy وخاصية `Icon` على moon.png

```
<FlyoutItem Title="Astronomy" Icon="moon.png">

</FlyoutItem>
```

٣- داخل `<FlyoutItem>` أضف `<ShellContent>` الذي يشير إلى `MoonPhasePage` اضبط خاصية `Title` الخاصة به على Moon Phase وخاصية `Icon` على moon.png

```
<FlyoutItem Title="Astronomy" Icon="moon.png">
  <ShellContent Title="Moon Phase" Icon="moon.png"
   ContentTemplate="{DataTemplate
local:MoonPhasePage}"/>
</FlyoutItem>
```

٤- ضمن نفس `<FlyoutItem>` أضف `<ShellContent>` آخر، للإشارة إلى `SunrisePage` اضبط خاصية `Title` الخاصة بها على Sunrise وخاصية `Icon` على sun.png

```
<FlyoutItem Title="Astronomy" Icon="moon.png">
  <ShellContent Title="Moon Phase" Icon="moon.png"
     ContentTemplate="{DataTemplate
local:MoonPhasePage}"/>

  <ShellContent Title="Sunrise" Icon="sun.png"
     ContentTemplate="{DataTemplate
local:SunrisePage}"/>
</FlyoutItem>
```

الآن، عند النقر على عنصر القائمة المنسدلة هذا، ستظهر صفحة علامة تبويب tab page تحتوي على علامتي تبويب two tabs

٥- لإنشاء عنصر منبثق جديد يشير إلى `AboutPage` أضف عنصرًا جديدًا `<FlyoutItem>` اضبط خاصية `Title` على About وخاصية `Icon` على question.png

٦- داخل `<FlyoutItem>` أضف `<ShellContent>` الذي يشير إلى `AboutPage`

```
<FlyoutItem Title="About" Icon="question.png">
   <ShellContent
```

```
      ContentTemplate="{DataTemplate
local:AboutPage}"/>
</FlyoutItem>
```

٧- شغّل التطبيق مرة أخرى. يجب أن تشاهد عنصرين في القائمة المنبثقة. يفتح الأول صفحة علامة تبويب تحتوي على `MoonPhasePage` and `SunrisePage` يعرض الثاني `AboutPage` بنفسه.

هل تحتاج مساعدة

يجب أن تبدو تعليمات XAML النهائية ل AppShell.xaml مثل هذا المثال:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
 x:Class="Astronomy.AppShell"
 xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
 xmlns:local="clr-namespace:Astronomy.Pages"
 FlyoutIcon="moon.png">

<!-- You can add this back in for the header -->
<!--<Shell.FlyoutHeader>
 <Grid HeightRequest="100"
BackgroundColor="DarkSlateBlue">
      <Image Source="moon.png"/>
 </Grid>
 </Shell.FlyoutHeader>-->

  <FlyoutItem Title="Astronomy" Icon="moon.png">
     <ShellContent Title="Moon Phase"
Icon="moon.png"
        ContentTemplate="{DataTemplate
local:MoonPhasePage}"/>

  <ShellContent Title="Sunrise" Icon="sun.png"
       ContentTemplate="{DataTemplate
local:SunrisePage}"/>
```

```
  </FlyoutItem>

  <FlyoutItem Title="About" Icon="question.png">
     <ShellContent
          ContentTemplate="{DataTemplate
local:AboutPage}"/>
  </FlyoutItem>
</Shell>
```

٦ استخدام الصفحات المبوبة tabbed pages مع الصفحات الموجودة في مجموعة التنقل the navigation stack

إن التنقل المنبثق والتنقل باستخدام علامات التبويب هما طريقتان لتمكين المستخدم من التنقل عبر البيانات التي يقدمها التطبيق، هناك تقنية أخرى تتمثل في استخدام التنقل بين الصفحات المجمعة أو المكدسة stack navigation يناسب كل نوع من أنواع التنقل، أنواعاً مختلفة من العلاقات بين الصفحات.

يمكنك استخدام التنقل بين الصفحات المجمعة stack navigation مع القوائم المنبثقة وعلامات التبويب flyouts and tabs في هذا الدرس، نستعرض بإيجاز الاختلافات بين التنقل عبر علامات التبويب والنوافذ المنبثقة والتنقل المكدس، ثم نتعلم كيفية إنشاء تطبيقات تجمع بين التقنيتين.

مراجعة القوائم المنبثقة flyouts والتنقل بين علامات التبويب tab navigation

توفر Flyouts and tab navigation آلية مفيدة لتمكين المستخدم من تحديد الصفحة التي تريد عرضها في أي وقت، ينتقل المستخدم إلى صفحة ما ببساطة عن طريق تحديد عنصر tab or flyout item المناسب، يعد هذا الشكل من التنقل مفيدًا لعناصر البيانات التي لها علاقة نظير مع بعضها البعض، في تطبيق علم الفلك the astronomy app تكون صفحات Sun, Moon, About pages جميعها نظيرات؛ ولا توجد علاقة مباشرة بينها.

ومع ذلك، بالنسبة للبيانات الهرمية، قد يكون التنقل بين الصفحات المجمعة stack navigation أكثر ملاءمة، يتيح التنقل بين الصفحات المجمعة للمستخدم التعمق في سلسلة من الصفحات، حيث توفر الصفحة التالية في المجموعة the next page in the stack عرضًا أكثر تفصيلاً لعنصر محدد في الصفحة السابقة.

فكر في السيناريو التالي:

يمكنك إضافة صفحة جديدة إلى تطبيق علم الفلك the astronomy تعرض الأجسام الفلكية، تريد أن ينقر المستخدم على أحد هذه الكائنات السماوية ثم تعرض معلومات عنها، الأفضل تقديم الصفحة التي تسرد الأجرام السماوية astronomical bodies في صفحة علامات تبويب tab page كصفحة شقيقة لصفحتي the moon phase and sunrise pages ومن الأفضل تقديم المعلومات التفصيلية حول الجسم الفلكي في سلسلة من الخطوات باستخدام التنقل المجمع stack navigation وتتيح هذه الآلية أيضًا للمستخدم العودة إلى صفحة الأجرام الفلكية الرئيسية the main astronomical bodies page بالنقر على زر back button

التنقل باستخدام تنقل الصفحات المجمعة Navigate with stack navigation

تتضمن واجهة مستخدم .NET Multi-platform App UI (MAUI) Shell تجربة تنقل مستندة إلى URI تستخدم المسارات للانتقال إلى أي صفحة في التطبيق، دون الحاجة إلى اتباع تسلسل هرمي محدد للتنقل، بالإضافة إلى ذلك، فإنه تتيح لك التنقل للخلف دون الحاجة إلى زيارة جميع الصفحات الموجودة في مجموعة التنقل the navigation stack

يحدد Shell class الخصائص التالية المتعلقة بالتنقل navigation-related

- `BackButtonBehavior` من النوع `BackButtonBehavior` خاصية مرفقة تحدد سلوك back button
- `CurrentItem` من النوع `ShellItem` العنصر المحدد حالياً.
- `CurrentPage` من النوع `Page` الصفحة المعروضة حالياً.
- `CurrentState` من النوع `ShellNavigationState` حالة التنقل الحالية لـ `Shell`
- `Current` من النوع `Shell` اسم مستعار مُحول حسب النوع type-casted alias لـ `Application.Current.MainPage`

يتم تنفيذ التنقل عن طريق استدعاء الأسلوب method `GoToAsync` من `Shell class`

مسارات Routes

يتم إجراء التنقل في تطبيق Shell من خلال تحديد عنوان URI للانتقال إليه، يمكن أن تحتوي عناوين URI للتنقل على ثلاث مكونات:

- **route** يحدد المسار إلى المحتوى، الموجود كجزء من التسلسل الهرمي المرئي لـ Shell
- **page** يمكن دفع الصفحات غير الموجودة في التسلسل الهرمي المرئي لـ Shell إلى مجموعة التنقل من أي مكان داخل تطبيق Shell على سبيل المثال، لا يتم تعريف صفحة التفاصيل في التسلسل الهرمي المرئي لـ Shell ولكن يمكن دفعها إلى مجموعة التنقل حسب الحاجة.
- معلمة استعلام **query parameters** واحدة أو أكثر، معلمات الاستعلام Query parameters هي المعلمات التي يمكن تمريرها إلى صفحة الوجهة أثناء التنقل.

عندما يتضمن عنوان URI للتنقل جميع المكونات الثلاثة، يكون الهيكل على النحو التالي: `//route/page?queryParameters`

تسجيل المسارات

يمكن تعريف المسارات Routes على `FlyoutItem, TabBar, Tab, ShellContent objects` من خلال خصائص `Route` الخاصة بهم:

```
<Shell ...>
    <FlyoutItem ...
        Route = "astronomy">
        <ShellContent ...
            Route="moonphase" />
        <ShellContent ...
            Route="sunrise" />
    </FlyoutItem>
    <FlyoutItem>
        <ShellContent ...
            Route="about" />
    </FlyoutItem>
</Shell>
```

للانتقال إلى `moonphase` route يكون route URI المطلق هو `//astronomy/moonphase`

تفاصيل تسجيل مسارات Register detail routes

في الأسلوب أو الدالة الإنشائية للفئة الفرعية Shell subclass constructor أو أي موقع آخر يتم تشغيله قبل استدعاء route يمكنك تسجيل register a route بشكل صريح باستخدام الأسلوب method `Routing.RegisterRoute` لأي صفحات تفاصيل غير ممثلة في التسلسل الهرمي المرئي لـ Shell

```
Routing.RegisterRoute("astronomicalbodydetails",
typeof(AstronomicalBodyPage));
```

للانتقال إلى `AstronomicalBodyPage` يجب عليك بعد ذلك باستدعاء:

```
await
Shell.Current.GoToAsync("astronomicalbodydetails");
```

التنقل للخلف Backwards navigation

يمكن إجراء التنقل للخلف Backwards navigation من خلال تحديد ".." كوسيطة argument لـ `GoToAsync` method

```
await Shell.Current.GoToAsync("..");
```

تمرير البيانات Passing data

يمكن تمرير البيانات الأولية كمعلمات استعلام تعتمد على سلسلة string-based query parameters عند تنفيذ التنقل البرمجي القائم على URI-based programmatic navigation مرر البيانات عن طريق إلحاق `?` بعد route متبوعًا بمعلمة الاستعلام query parameter ID ثم `=` والـ value

```
string celestialName = "moon";

await
Shell.Current.GoToAsync($"astronomicalbodydetails?
bodyName={celestialName}");
```

في هذا المثال، المسار route هو astronomicalbodydetails والمعلمة parameter هي bodyName والقيمة value من المتغير the variable `celestialName`

استرداد البيانات Retrieving data

يمكن استقبال بيانات التنقل من خلال تزيين فئة الاستقبال بـ `QueryPropertyAttribute` لكل معلمة استعلام تعتمد على سلسلة string-based query parameter ومعلمة تنقل تعتمد على كائن object-based navigation parameter

```
[QueryProperty(nameof(AstroName), "bodyName")]
public partial class AstronomicalBodyPage : ContentPage
{
    string astroName;
    public string AstroName
    {
        get => astroName;
        set
        {
            astroName = value;
        }
```

```
    }
    ...
}
```

في هذا المثال، تحدد الوسيطة argument الأولى لـ QueryPropertyAttribute اسم الخاصية التي تتلقى البيانات، بينما تحدد الوسيطة الثانية معرف المعلمة the parameter ID

اختبار بسيط:

١- كيف تقوم بإرسال البيانات إلى صفحة عند استخدام التنقل القائم على المسار .NET MAUI Shell route-based navigation.

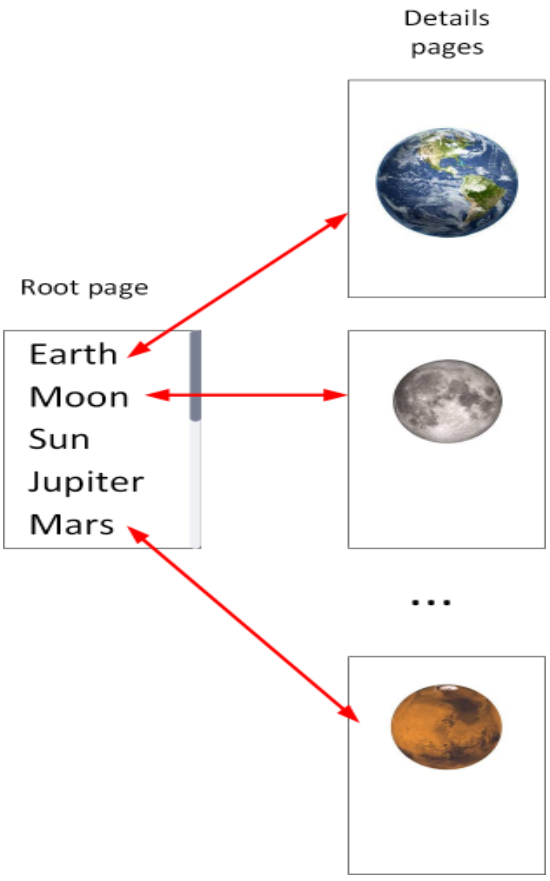
حل الاختبار:

١- استخدم بناء جملة سلسلة استعلام query string syntax يمكن إرسال زوج key/value

يمكنك استخدام بناء جملة سلسلة استعلام query string syntax لإرسال البيانات بين الصفحات send data between pages

٧ تمرين: استخدام الصفحات المبوبة tabbed pages مع صفحات التنقل navigation pages

في تطبيق علم الفلك the astronomy يُطلب منك إضافة صفحات تمكن المستخدم من تحديد أجسام فلكية مختلفة وعرض تفاصيلها، يمكن أن يكون هناك أي عدد من الأجسام، لذا فإن إنشاء علامة تبويب لكل جسم ليس نهجًا عمليًا، لذلك، لتمكين المستخدم من تحديد الجسم الذي سيتم عرضه، تقرر إضافة صفحة أخرى تحتوي على قائمة. يمكن للمستخدم تحديد جسم من هذه القائمة ويعرض التطبيق تفاصيل هذا الجسم في صفحة جديدة، تعمل صفحة القائمة كصفحة جذر للتنقل بين الصفحات المجمعة، يمكنك إضافة صفحة القائمة كعلامة تبويب في واجهة المستخدم الحالية.



افتح the starter solution

١- انتقل إلى مجلد exercise3 في المستودع الذي نسخته في بداية هذه الوحدة، ثم انتقل إلى مجلد start

٢- استخدم Visual Studio لفتح Astronomy.sln solution أو المجلد في Visual Studio Code

ملحوظة: يحتوي هذا solution على صفحات وأنماط pages and styles غير موجودة في إصدار التطبيق الذي استخدمته في التمارين السابقة.

٣- في نافذة مستكشف الحلول Solution Explorer افتح المجلد Pages بالإضافة إلى ملفات MoonPhasePage, SunrisePage, AboutPage files يحتوي هذا المجلد على صفحتين إضافيتين:

- AstronomicalBodiesPage تحتوي هذه الصفحة على أربعة أزرار تمكن المستخدم من تحديد تفاصيل الشمس أو القمر أو الأرض أو مذنب هالي the Sun, the Moon, the Earth, or Halley's Comet الإصدار الحالي من التطبيق هو مجرد دليل على المفهوم. في المستقبل، تمكن هذه الصفحة المستخدم من الاختيار من قائمة أكبر.
- AstronomicalBodyPage تُستخدم هذه الصفحة لعرض المعلومات الخاصة ب astronomical body

تمت إضافة AstronomicalBodiesPage بالفعل كعلامة تبويب على الصفحة التي تظهر عند النقر فوق القائمة المنبثقة Astronomy flyout

إضافة route لصفحة التفاصيل the details page

١- للانتقال إلى AstronomicalBodyPage تحتاج إلى إعداد route عن طريق تسجيله في الدالة الإنشائية لفئة AppShell class

```
public AppShell()
{
    InitializeComponent();

  Routing.RegisterRoute("astronomicalbodydetails",
typeof(AstronomicalBodyPage));
}
```

الانتقال إلى صفحة التفاصيل Navigate to the details page

١- الآن نحتاج إلى تنفيذ عملية التنقل، في AstronomicalBodiesPage.xaml.cs قم بإنشاء معالجات حدث النقر click-event handlers لكل Button في الصفحة.

```
public AstronomicalBodiesPage()
{
    InitializeComponent();

    btnComet.Clicked += async (s, e) => await
Shell.Current.GoToAsync("astronomicalbodydetails");
    btnEarth.Clicked += async (s, e) => await
Shell.Current.GoToAsync("astronomicalbodydetails");
```

```
    btnMoon.Clicked += async (s, e) => await
Shell.Current.GoToAsync("astronomicalbodydetails");
    btnSun.Clicked += async (s, e) => await
Shell.Current.GoToAsync("astronomicalbodydetails");
}
```

عند النقر على أي Button ينتقل التطبيق إلى صفحة AstronomicalBodyPage ولكننا ما زلنا بحاجة إلى إرسال نوع الجسم الفلكي type of astronomical body المراد عرضه.

٢- لإرسال البيانات إلى AstronomicalBodyPage أضف سلسلة معلمات الاستعلام query parameter string إلى معلومات التوجيه the routing information من النموذج The string السلسلة `?astroName=astroBodyToDisplay`

```
btnComet.Clicked += async (s, e) => await
Shell.Current.GoToAsync("astronomicalbodydetails?astroName
=comet");
btnEarth.Clicked += async (s, e) => await
Shell.Current.GoToAsync("astronomicalbodydetails?astroName
=earth");
btnMoon.Clicked += async (s, e) => await
Shell.Current.GoToAsync("astronomicalbodydetails?astroName
=moon");
btnSun.Clicked += async (s, e) => await
Shell.Current.GoToAsync("astronomicalbodydetails?astroName
=sun");
```

٣- لتلقي البيانات على AstronomicalBodyPage قم أولاً بإنشاء خاصية على مستوى الفئة class-level property للاحتفاظ بالبيانات الواردة، قم بتسميتها `AstroName`

```
string astroName;
public string AstroName
{
    get => astroName;
    set
    {
        astroName = value;
```

```
// this is a custom function to update the UI immediately
      UpdateAstroBodyUI(astroName);
    }
}
```

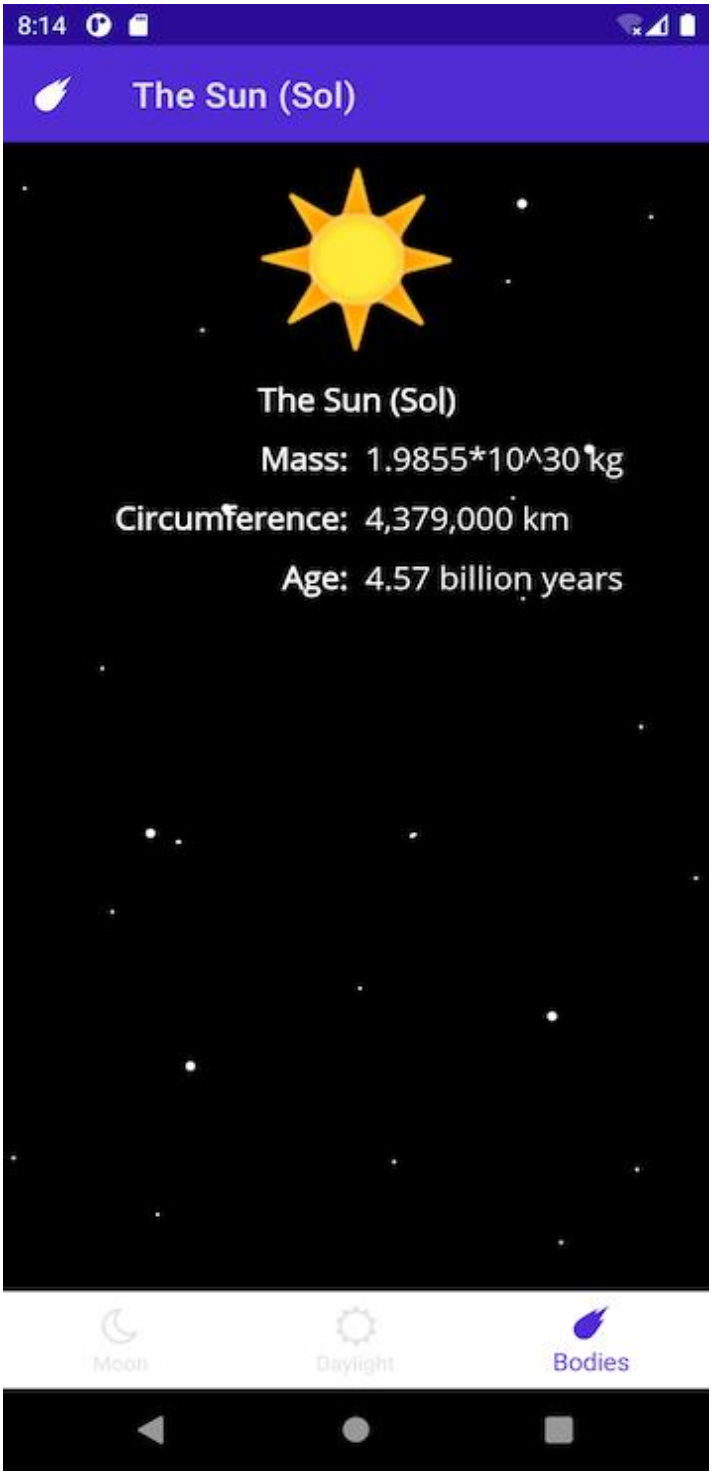
هنا، يعد `UpdateAstroBodyUI(astroName)` عبارة عن دالة مساعدة helper function تستخدم لتحديث واجهة المستخدم على الفور عند تعيين خاصية `AstroName`

٤- عليك تزيين الفئة decorate the class مع تعليق توضيحي يربط معلمة الاستعلام الواردة incoming query parameter بالخاصية التي قمت بإنشائها.

```
[QueryProperty(nameof(AstroName), "astroName")]
public partial class AstronomicalBodyPage
{
    ...
}
```

٥- ابدأ تشغيل التطبيق وحدد علامة التبويب بعنوان Bodies

٦- حدد Sun button يجب أن تظهر تفاصيل الشمس The details for the Sun يجب أن يحتوي شريط التنقل The navigation bar على سهم للخلف back arrow للسماح للمستخدم بالعودة إلى the list of bodies تظل علامات التبويب المتبقية مرئية ونشطة:



٧- أغلق التطبيق وارجع إلى Visual Studio أو Visual Studio Code

٨ الملخص

يعد تحديد كيفية تنقل المستخدم بين الصفحات جزءًا من تخطيط بنية التطبيق، في هذه الوحدة، تعلمت كيفية استخدام القوائم المنبثقة flyouts والصفحات المبوبة tabbed pages لتنفيذ تطبيق يعرض البيانات على صفحات متعددة، يجب أن تفهم الآن متى يكون نمط التنقل هذا مناسبًا.

تعلمت كيفية تنفيذه في واجهة مستخدم التطبيق متعددة الأنظمة .NET Multi-platform App UI (MAUI) وكيف يمكنك تخصيص سلوك الصفحات المبوبة the behavior of tabbed pages تعلمت أيضًا كيفية دمج الصفحات المبوبة مع صفحات التنقل navigation pages التي تستخدم مكدس التنقل the navigation stack

تعلمت عَلى وجه التحديد كيفية القيام بما يلي:

- تنفيذ التنقل المنبثق flyout navigation باستخدام .NET MAUI Shell
- تنفيذ تنقل علامات التبويب tab navigation باستخدام .NET MAUI Shell
- التنقل بين الصفحات داخل الصفحات المبوبة tabbed pages

الوحدة السادسة

إنشاء واجهة مستخدم UI تستخدم ربط البيانات data binding في .NET MAUI

تعرف على كيفية إضافة ربط البيانات إلى واجهة المستخدم الخاصة بك. باستخدام ربط البيانات، يتم تحديث واجهة المستخدم تلقائيًا عند تغير البيانات. عندما لا تتطابق أنواع البيانات، يمكنك استخدام محول لتحويل البيانات لواجهة المستخدم.

الأهداف التعليمية

في نهاية هذه الوحدة، ستتمكن من:

- قارن بين واجهة مستخدم مدفوعة بالأحداث event-driven UI وواجهة مستخدم مرتبطة بالبيانات data-bound UI
- إنشاء كائنات بيانات قابلة للاستخدام data objects usable مع ربط البيانات data binding
- تصميم واجهة مستخدم مدفوعة بربط البيانات driven by data binding

محتويات الوحدة:

١- المقدمة

٢- مقارنة بين واجهة مستخدم مرتبطة بالأحداث event-driven UI وواجهة مستخدم مرتبطة بالبيانات data-bound UI

٣- استخدام ربط البيانات data bindings في XAML

٤- تمرين - استبدال التعليمات بروابط .NET MAUI bindings

٥- ربط البيانات Data binding مع أنواع غير متطابقة mismatched types

٦- تمرين - إنشاء محول .NET MAUI converter

٧- الملخص

١ المقدمة

يتيح لك ربط البيانات Data binding إعلان العلاقات relationships بين بياناتك وعناصر التحكم في واجهة المستخدم (UI) تظل هذه العناصر متزامنة sync تلقائيًا عند تغير أي من العنصرين، تجعل ارتباطات البيانات Data bindings التعليمات البرمجية أسرع في القراءة، وأسهل في الصيانة، وأكثر قابلية للاختبار.

يوفر ربط البيانات طريقة لربط واجهة المستخدم بالبيانات مباشرةً، بدلاً من ربط واجهة المستخدم الخاصة بك بالتعليمات الموجود في التعليمات البرمجية الخلفية لواجهة المستخدم، عندما يتم ربط واجهة المستخدم بالبيانات، يتم تحديث واجهة المستخدم تلقائيًا عند تغير البيانات، وتبقى متزامنة مع البيانات، وفي المقابل، إذا تفاعل المستخدم مع واجهة المستخدم، يتم الحفاظ على البيانات متزامنة مع واجهة المستخدم.

سيناريو مثال

تخيل أنك تعمل على تطبيق يقدم للمستخدمين معلومات الطقس لمنطقة أو عنوان محدد، بينما يقوم المستخدم بإدخال موقع، يقوم التطبيق بتحميل بيانات التنبؤ بالطقس من خدمة طقس خارجية، وتحديث واجهة المستخدم. تقترن واجهة المستخدم وبيانات خدمة الطقس بشكل وثيق عبر واجهة مستخدم XAML وملف XAML الذي يحتوي على التعليمات البرمجية، تم بالفعل تجريد خدمة الطقس إلى فئتها الخاصة، ولكن واجهة المستخدم الخاصة بك تعتمد بشكل كبير على التعليمات البرمجية الخلفية لتقديم تلك البيانات إلى واجهة المستخدم.

بعد مرور بعض الوقت على إصدار التطبيق، تعلن خدمة الطقس التي تستخدمها أنها ستتوقف عن العمل، وتوقعًا لذلك، تبدأ في البحث عن خدمة بديلة، ومن بين الخدمات التي قمت بتقييمها، لاحظت أنها جميعًا تقوم بإدخال البيانات وإخراجها بطرق مختلفة. ونظرًا لأن واجهة المستخدم الخاصة بك تستخدم أحداثًا للتعليمات البرمجية الخلفية code-behind events للتفاعل مع الخدمة، فقد تتأثر تجربة واجهة المستخدم الخاصة بك بهذا التغيير.

سيكون من المفيد تغيير كيفية مزامنة واجهة المستخدم لتحديثات الطقس، من أحداث التعليمات البرمجية الخلفية code-behind events إلى ربط البيانات data binding بهذه الطريقة، إذا حدث تغيير في الخدمة مرة أخرى، فقد قللت من التأثير على التعليمات البرمجية.

ماذا ستفعل

توضح هذه الوحدة الاختلافات بين واجهة المستخدم مرتبطة بالتعليمات البرمجية code-based UI وواجهة مستخدم مرتبطة بالبيانات data-bound UI وتوضح لك كيفية استخدام ربط البيانات data binding لتحديث واجهة المستخدم بدلاً من التعليمات البرمجية الخلفية code-behind باستخدام نموذج تطبيق الطقس، ستقوم بتحويل تحديثات واجهة المستخدم من التعليمات البرمجية خلفية إلى ربط البيانات.

ما الذي تتعلمه

بنهاية هذه الوحدة، يمكنك استخدام ربط البيانات لتقديم البيانات في واجهة المستخدم، وتحويل البيانات عندما لا تتطابق أنواع واجهة المستخدم UI types وأنواع البيانات data types

٢ مقارنة بين واجهة مستخدم مرتبطة بالأحداث event-driven UI وواجهة مستخدم مرتبطة بالبيانات data-bound UI

تم تصميم واجهة مستخدم مرتبطة بالأحداث أو تستند إلى الأحداث event-driven user interface (UI) حول الأحداث التي يكشفها عنصر التحكم، يمكن ربط هذه الأحداث بتعليمات معالج الأحداث التي يتم استدعاؤها عند تشغيل الحدث، على سبيل المثال، لنفترض أن لديك زراً يقوم عند النقر فوقه بإجراء عملية طويلة الأمد long-running operation يمكن لمعالج الأحداث المعين للحدث Clicked بدء العملية ثم تعيين خاصية الزر IsEnabled إلى false مما يمنع النقر فوق الزر مرة أخرى أثناء تشغيل العملية.

تستخدم واجهة المستخدم المرتبطة بالبيانات data-bound UI ربط البيانات data binding لتقديم البيانات والتفاعل معها، ترتبط خصائص عناصر التحكم بخصائص كائن البيانات data object's ويمكن لهذه الارتباطات اكتشاف التغييرات في الخصائص. باستخدام المثال السابق، ضع في اعتبارك الزر الذي ينفذ عملية طويلة الأمد long-running operation بدلاً من تعطيل الزر في التعليمات البرمجية الخلفية، يتم ربط الخاصية IsEnabled بخاصية IsBusy الخاصة بكائن البيانات، كلما أصبح كائن البيانات "busy" يتم تغيير حالة button's enabled تلقائياً لمطابقتها.

إيجابيات وسلبيات استخدام الأحداث events والتعليمات الخلفية code-behind

يعد استخدام معالج أحداث عنصر التحكم control's event handler مع التعليمات الخلفية طريقة سريعة ومريحة لتصميم منطق التطبيق لواجهة المستخدم، يمكنك استخدام التعليمات البرمجية لاستدعاء الخدمات للحصول على البيانات، وتنفيذ العمليات على تلك البيانات، والتفاعل مع عناصر التحكم الموجودة على الصفحة، يتم استخدام التعليمات البرمجية للحفاظ على مزامنة واجهة المستخدم والبيانات.

ضع في اعتبارك مثال تطبيق خدمة الطقس، يحتوي جزء XAML التالي على زر واجهة مستخدم بسيط يحدده المستخدم للحصول على أحدث البيانات وتحديث واجهة المستخدم بالرطوبة humidity

```
<VerticalStackLayout Margin="10">
  <HorizontalStackLayout Spacing="20">
      <Label Text="Postal Code:"
VerticalOptions="Center" />
     <Entry x:Name="PostalCode" WidthRequest="100" />
      <Button x:Name="RefreshWeatherButton"
Text="Refresh" WidthRequest="200"
Clicked="RefreshWeatherButton_Clicked" />
```

```
  </HorizontalStackLayout>
  <Label x:Name="Humidity" Text="Humidity: ?" />
</VerticalStackLayout>
```

هناك ثلاثة عناصر تحكم مسماة في هذا المثال:

Entry المسمى **PostalCode**

Button المسمى **RefreshWeatherButton**

Label المسمى **Humidity**

يحتوي زر RefreshWeatherButton على معالج حدث معلن لحدث Clicked عند النقر فوق الزر، يستعلم معالج الحدث عن توقعات الطقس من خدمة الطقس، باستخدام البيانات المدخلة في عنصر التحكم PostalCode entry ويضبط نص Humidity label على الرطوبة الحالية.

```
private void RefreshWeatherButton_Clicked(object
sender, EventArgs e)
{
   WeatherService.Location = PostalCode.Text;
   WeatherService.Refresh();
   Humidity.Text = $"Humidity:
{WeatherService.Humidity}";
}
```

في معالج الأحداث هذا، تقترن ثلاثة عناصر تحكم بإحكام ببعضها البعض وبالبيانات من خلال التعليمات البرمجية الخلفية.

يعمل هذا التصميم بشكل رائع مع واجهات المستخدم الصغيرة، ولكن بمجرد أن تصبح واجهة المستخدم معقدة، ربما يصبح الحفاظ على التعليمات الخلفية المرتبطة بإحكام أمرًا مزعجًا، إذا قمت بحذف أو تغيير عنصر تحكم، يجب عليك تنظيف أي تعليمات باستخدام عناصر تحكم واجهة المستخدم هذه، والتي قد تتضمن معالج الحدث، إذا قررت إعادة تصميم واجهة المستخدم، فسيكون لديك الكثير من التعليمات لإعادة صياغتها أيضًا، وعندما تتغير بنية بيانات النسخ الاحتياطي the backing data structure يجب عليك التعمق في التعليمات لكل واجهة مستخدم للبقاء في وضع المزامنة stay in sync

يساعد ربط البيانات Data binding helps

يمكن تنفيذ ربط البيانات Data bindings في XAML أو التعليمات البرمجية، ولكنها أكثر شيوعاً في XAML حيث تساعد على تقليل حجم ملف التعليمات البرمجية الخلفية، من خلال استبدال التعليمات البرمجية الإجرائية في معالجات الأحداث event handlers بتعليمات تعريفية أو علامات declarative code or markup يتم تبسيط التطبيق وتوضيحه، نظرا لأن الارتباطات bindings لا تتطلب تعليمات برمجية في الخلف، يمكنك بسهولة إنشاء واجهة المستخدم أو تغييرها أو إعادة تصميمها لتناسب الطريقة التي تريد بها تقديم البيانات.

لنأخذ نفس المثال كما في القسم السابق، ولكن قم بتحديثه لاستخدام ربط البيانات:

```
<VerticalStackLayout Margin="10">
  <HorizontalStackLayout Spacing="20">
    <Label Text="Postal Code:" VerticalOptions="Center" />
      <Entry Text="{Binding Location,
Mode=OneWayToSource}" WidthRequest="100" />
      <Button Text="Refresh" Command="{Binding
RefreshWeather}" WidthRequest="200" />
  </HorizontalStackLayout>
  <Label Text="{Binding Humidity}" />
</VerticalStackLayout>
```

يمكنك تحديد الخصائص المرتبطة بالبيانات، وهي تستخدم بناء جملة {Binding ...} ملحق XAML لقيمة الخاصية XAML extension syntax {Binding ...} for the value of the property لا تقلق بشأن التفاصيل حتى الآن، سيتم تناولها لاحقاً في هذه الوحدة.

يتم تعريف عناصر التحكم الثلاثة نفسها في XAML ولكن لا يتم تسمية أي منها، لأن الاسم غير مطلوب:

- `Entry` control

خاصية `Text` الخاصة بعنصر التحكم، مرتبطة بخاصية تسمى `Location`

- `Button` control

ترتبط خاصية `Command` الخاصة بالزر بخاصية تسمى `RefreshWeather` ال Command هي خاصية في الزر تستدعي التعليمات البرمجية عند الضغط على الزر، هي بديل لحدث `Clicked` المستخدم في ربط البيانات data binding

- `Label` control

ترتبط خاصية `Text` بخاصية تسمى `Humidity`

في واجهة المستخدم البسيطة هذه، يتم التخلص من كل التعليمات البرمجية الخلفية، لا يعد إزالة كل التعليمات البرمجية الخلفية هو الهدف من ربط البيانات، على الرغم من أنه ممكن عادةً، لا يزال للتعليمات البرمجية الخلفية أهميتها، مقدار ربط البيانات الذي تنفذه متروك لك.

الآن واجهة المستخدم مرتبطة بشكل غير محكم بكائن البيانات data object لماذا تم ربطها بشكل غير محكم بدلاً من ربطها بإحكام؟ بسبب الطريقة التي يتم بها تقييم الارتباطات، يحتوي كل عنصر تحكم على خاصية `BindingContext` إذا لم يتم تعيين السياق، يتم استخدام سياق عنصر التحكم الأصل، وهكذا، حتى يتم تقييم جذر XAML عند تقييم الارتباطات، يتم التحقق من مثيل كائن السياق context's object بحثاً عن الخصائص المطلوبة، مثل ارتباط النص `Text` binding الخاص بعنصر التحكم label بخاصية `Humidity` في السياق، إذا لم تكن `Humidity` موجودة في السياق، فلن يحدث أي شيء.

نظرًا لأن واجهة المستخدم غير مترابطة بشكل غير محكم، يمكنك إعادة تصميم واجهة المستخدم دون القلق بشأن كسر التعليمات البرمجية breaking code ومع ذلك، يمكنك كسر الوظائف break functionality على سبيل المثال، يمكنك حذف الزر ويظل التطبيق قيد التجميع والتشغيل، ولكن ليس لديك طريقة لتحديث الطقس refresh the weather من ناحية أخرى، يمكنك استبدال عناصر التحكم `Entry` and `Button` بعنصر تحكم `SearchBar` واحد. يتيح لك عنصر التحكم هذا إدخال نص واستدعاء أمر enter text and invoke a command

```
<SearchBar Text="{Binding Location, Mode=OneWayToSource}"
SearchCommand="{Binding RefreshWeather}" />
```

كما ترى، فإن استخدام ربط البيانات data binding في تصميم واجهة المستخدم الخاصة بك يمكن أن يساعدك على تطوير واجهة المستخدم، وتغييرها دون بذل الكثير من الجهد، فهو يحافظ على مزامنة واجهة المستخدم مع البيانات تلقائيًا، ويتم فصل منطق التطبيق عن واجهة المستخدم.

اختبار بسيط:

١- أي من العبارات التالية تصف كيفية تحسين ربط البيانات data binding improves لواجهة المستخدم؟

- لن تضطر أبدا إلى كتابة التعليمات البرمجية المتعلقة بواجهة المستخدم الخاصة بك مرة أخرى.
- يتم مزامنة واجهة المستخدم تلقائيًا مع البيانات.

٢- في تطبيق يسترجع أحدث حالة الطقس، يرتبط عنصر label الذي يمثل درجة الحرارة temperature بكائن البيانات، عندما يضغط المستخدم الزر "Get Weather" button مما يؤدي إلى تشغيل الحدث Clicked ماذا ستفعل التعليمات البرمجية لمعالج الحدث لتحديث واجهة المستخدم؟

- من معالج الحدث، قم باستدعاء خدمة الطقس weather service وتحديث كائن بيانات الطقس weather data object بأحدث المعلومات.
- من معالج الحدث، لا تفعل شيئًا، إن temperature label مرتبط بالبيانات، لذا يجب تحديثه تلقائيًا.
- من معالج الحدث، قم باستدعاء خدمة الطقس weather service ثم اضبط نص ملصق درجة الحرارة temperature label's text

حل الاختبار:

١- يتم مزامنة واجهة المستخدم تلقائيًا مع البيانات.

من فوائد إضافة ربط البيانات إلى واجهة المستخدم، أنها تحافظ على مزامنة عناصر واجهة المستخدم مع خصائص كائن البيانات. كما أنها تساعد في فصل منطق التطبيق عن عرض التطبيق التقديمي.

٢- من معالج الحدث، قم باستدعاء خدمة الطقس weather service وتحديث كائن بيانات الطقس weather data object بأحدث المعلومات.

نظراً لأن التعليمات البرمجية الخلفية تتحدث "talking" فقط إلى كائن البيانات data object وليس Label وهو غير مرتبط بإحكام ب Label يقوم Label بتحديث نصه تلقائيًا عند تغيير كائن البيانات المرتبط.

٣ استخدام ربط البيانات data bindings في XAML

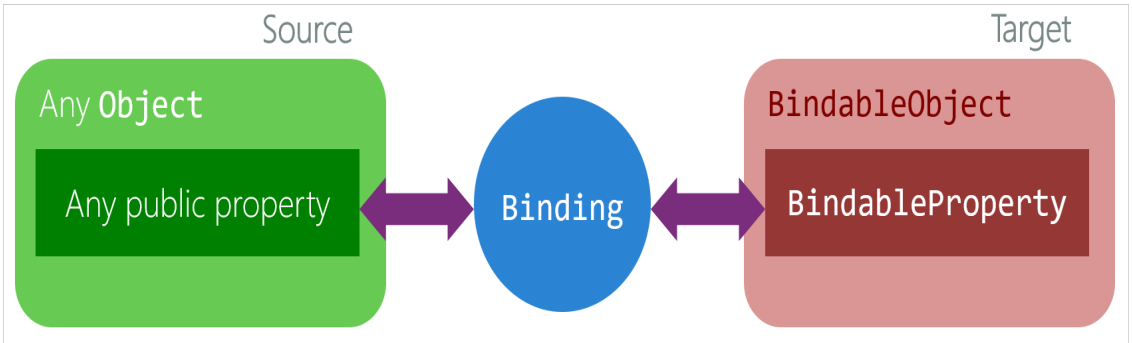
يمكن تعريف روابط البيانات Data bindings إما في التعليمات البرمجية أو في XAML باستخدام ملحقات العلامات markup extensions يناقش هذا الدرس الطريقة الأخيرة لأنها الطريقة الأكثر شيوعًا لإنشاء الروابط. هناك سببان لتفضيل XAML أولاً، يعتبر معظم الأشخاص أن الارتباطات جزء من تعليمات واجهة المستخدم الخاصة بهم، لأن الارتباطات تحصل على بيانات لعرضها في واجهة المستخدم، ثانيًا، هناك ملحق علامات اسمه `Binding` يسهل القيام بذلك.

ما هي روابط البيانات data bindings

يعمل ربط البيانات على ربط خاصيتين اثنتين معًا، خاصية واحدة في واجهة المستخدم الخاصة بك والأخرى في كائن نموذج البيانات data-model object إذا تغيرت قيمة أي خاصية، يمكن لكائن الربط binding object تحديث الخاصية الأخرى، بعبارة أخرى، الروابط عبارة عن كائنات وسيطة bindings are intermediary objects تعمل على مزامنة واجهة المستخدم والبيانات UI and data نستخدم مصطلحي المصدر source والهدف target لتحديد الكائنين المعنيين:

- المصدر source يمكن أن يكون المصدر كائنًا من أي نوع object of any type في الممارسة العملية، تستخدم عادةً كائن بيانات data-object كمصدر source لك، تحتاج إلى تعريف الخاصية في هذا الكائن المصدر source object للمشاركة في الربط. يمكنك تحديد الخاصية عن طريق تعيين الخاصية `Path` في الربط.
- الهدف Target هو خاصية يتم تنفيذها باستخدام خاصية خاصة تسمى `BindableProperty` يجب أن يكون الكائن الذي يحتوي `BindableProperty` مشتقًا من `BindableObject` جميع عناصر التحكم المتوفرة في .NET MAUI. مشتقة من `BindableObject` ومعظم خصائصها هي `BindableProperties`

يوضح الرسم التخطيطي التالي كيف أن الربط هو كائن وسيط binding is an intermediary object بين خاصيتين:



كيفية إنشاء ربط بيانات data binding في XAML

لنلق نظرة على رابط بسيط تم إنشاؤه في XAML باستخدام ملحق العلامات `{Binding}` وهو يربط خاصية `WeatherService.Humidity` للمصدر بخاصية `Text` لعنصر التحكم في واجهة المستخدم.

```
<VerticalStackLayout Margin="10">
  <VerticalStackLayout.Resources>
     <ResourceDictionary>
        <services:WeatherService
x:Key="myWeatherService" />
     </ResourceDictionary>
  </VerticalStackLayout.Resources>

  <Label Text="{Binding Path=Humidity,
Source={StaticResource myWeatherService}}" />
</VerticalStackLayout>
```

مصدر الربط The binding source هو:

- مثيل كائن من النوع `WeatherService` تتم الإشارة إلى المثيل من خلال XAML extension `{StaticResource ...}` الذي يشير إلى كائن في قاموس موارد تخطيط المكدس stack layout's resource dictionary
- يشير `Path` إلى خاصية تسمى `Humidity` في النوع `WeatherService`

`Path` هو أول معلمة غير مسماة في بناء الجملة `{Binding}` ويمكن حذف بناء الجملة `Path=` هذان الارتباطان متكافئان:

```
<Label Text="{Binding Path=Humidity,
Source={StaticResource myWeatherService}}" />
<Label Text="{Binding Humidity, Source={StaticResource
myWeatherService}}" />
```

هدف الربط هو:

- عنصر التحكم Label
- خاصية عنصر التحكم The control's `Text` property

عند عرض واجهة المستخدم، يقوم ملحق علامات `{Binding}` XAML extension بإنشاء ارتباط بين `Label` and `WeatherService` يقرأ

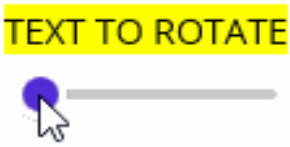
الارتباط قيمة خاصية `WeatherService.Humidity` في خاصية `Label.Text`

استخدام عنصر تحكم آخر another control كمصدر ربط binding source

إحدى الميزات المفيدة للربط هي القدرة على الربط بعناصر التحكم الأخرى، مثال XAML التالي هو عرض توضيحي بسيط:

```
<VerticalStackLayout HorizontalOptions="Center"
VerticalOptions="Center">
  <Label x:Name="TargetLabel" Text="TEXT TO ROTATE"
BackgroundColor="Yellow" />
  <Slider WidthRequest="100" Maximum="360"
      Value="{Binding Rotation, Mode=OneWayToSource,
Source={x:Reference TargetLabel}}" />
</VerticalStackLayout>
```

ترتبط خاصية `Slider.Value` بخاصية `Label.Rotation` ولكن بطريقة مختلفة عن التي تم شرحها سابقًا، تستخدم هذه الخاصية وضع الربط `OneWayToSource` الذي يعكس آلية الربط النموذجية، فبدلاً من تحديث `Source` لـ `Target` يقوم `OneWayToSource` بتحديث Source عند تغير `Target` في هذا المثال، عندما يتحرك شريط التمرير، فإنه يقوم بتحديث دوران الملصق the rotation of the label بناءً على قيمة شريط التمرير slider's value كما هو موضح في الرسوم المتحركة التالية:



السيناريو النموذجي لربط عناصر التحكم ببعضها البعض هو عندما يكون هناك عنصر تحكم، عادةً ما يكون عنصر التحكم في مجموعة مثل `ListView` or `CarouselView` يحتوي على عنصر محدد تريد استخدامه كمصدر بيانات data source في مثال الصفحة التي تعرض توقعات الطقس، قد يكون لديك ListView يعرض توقعات الطقس لمدة خمسة أيام، عندما يختار المستخدم يومًا في القائمة، يتم عرض تفاصيل توقعات الطقس هذه في عناصر تحكم أخرى، إذا

اختار المستخدم يومًا آخر، يتم تحديث عناصر التحكم الأخرى مرة أخرى بتفاصيل اليوم المحدد.

استخدام نفس المصدر عبر روابط متعددة multiple bindings

أوضح المثال السابق استخدام مورد ثابت static resource كمصدر لربط واحد source for a single binding ويمكن استخدام هذا المصدر في روابط متعددة multiple bindings فيما يلي مثال للإعلان عن ربط عبر ثلاثة عناصر تحكم مختلفة، جميعها مرتبطة بنفس الكائن object والخاصية `Path` على الرغم من أن بعضها يحذف الخاصية `Path`

```
<VerticalStackLayout Margin="10">
 <VerticalStackLayout.Resources>
    <vm:SimpleWeatherServiceObject
x:Key="myWeatherService" />
 </VerticalStackLayout.Resources>
 <Entry Text="{Binding Humidity,
Source={StaticResource myWeatherService}}" />
 <Label Text="{Binding Path=Humidity,
Source={StaticResource myWeatherService}}" />
</VerticalStackLayout>
```

لا يتعين عليك استخدام نفس المسار `Path` عند استخدام نفس المصدر `Source`

```
<VerticalStackLayout Margin="10">
 <VerticalStackLayout.Resources>
     <vm:SimpleWeatherServiceObject
x:Key="myWeatherService" />
 </VerticalStackLayout.Resources>
 <Entry Text="{Binding Temperature,
Source={StaticResource myWeatherService}}" />
 <Label Text="{Binding Path=Humidity,
Source={StaticResource myWeatherService}}" />
</VerticalStackLayout>
```

نادرًا ما تقوم بتقديم جزءاً واحدًا من بيانات واحدة من مصدر واحد، على الرغم أنه قد يحدث، عادةً ما يكون لديك عدة عناصر تحكم تستخدم أجزاء بيانات مختلفة من نفس المصدر. هذا الموقف شائع جدًا لدرجة أن class `BindableObject` تحتوي على خاصية تسمى `BindingContext` تعمل كمصدر لربط البيانات source for data binding تذكر أن عناصر تحكم .NET MAUI ترث من فئة

`BindableObject` class لذا فإن عناصر تحكم .NET MAUI. تحتوي على خاصية `BindingContext`

يعد تعيين مصدر الربط اختياريًا، يبحث تلقائيًا الربط الذي لا يحتوي على مصدر معيين، في شجرة XAML المرئية عن `BindingContext` والذي يتم تعيينه في XAML أو تعيينه إلى عنصر أصل بواسطة التعليمات، يتم تقييم الارتباطات وفقًا لهذا النمط:

١- إذا كان الربط يعرف Source يتم استخدام هذا المصدر، ويتوقف البحث. يتم تطبيق مسار الربط binding's Path على Source للحصول على قيمة get a value إذا لم يتم تعيين Source يبدأ البحث عن مصدر ربط binding source

٢- يبدأ البحث بالكائن المستهدف the target object نفسه، إذا لم يكن BindingContext الخاص بالكائن المستهدف فارغًا null يتوقف البحث، ويتم تطبيق مسار الربط binding's Path على BindingContext للحصول على قيمة get a value إذا كان `BindingContext is null` فارغًا، يستمر البحث.

٣- تستمر هذه العملية حتى تصل إلى جذر XAML root ينتهي البحث بفحص BindingContext للجذر بحثًا عن قيمة غير فارغة non-null value إذا لم يتم العثور على BindingContext صالح، فلن يكون للربط أي شيء يرتبط به ولن يفعل أي شيء.

من الشائع تعيين BindingContext على مستوى الكائن الجذر the root object's level لتطبيقه على XAML بالكامل

ميزة ملائمة أخيرة تستحق الذكر، تراقب الروابط Bindings التغييرات التي تطرأ على مرجع الكائن object reference الخاص بمصدرها source وهذا ينطبق حتى على الروابط التي تستخدم BindingContext كمصدر لها. إذا تم إعادة تعيين source أو BindingContext إلى كائن آخر، فإن الروابط تلتقط البيانات من المصدر الجديد وتقوم بتحديث الهدف target

اختبار بسيط:

١- هو الصحيح بالنسبة لكائن المصدر source object في .NET MAUI binding

- يمكن أن يكون أي نوع any type
- يجب أن يكون BindableObject
- يجب أن يكون فئة فرعية subclass من View

٢- ما هو الصحيح بالنسبة لخاصية target في .NET MAUI binding

- يمكن أن تكون public property
- يجب أن تكون BindableProperty
- يجب أن تكون DependencyProperty

٣- إذا كانت جميع الارتباطات bindings الموجودة على عناصر التحكم داخل Grid تحتاج إلى نفس كائن المصدر source object فما هي الاستراتيجية الأكثر أمانًا لتعيين كائن المصدر مرة واحدة فقط؟

حل الاختبار:

١- يمكن أن يكون كائن المصدر source object أي نوع any type

٢- يجب أن تكون خاصية target كائن BindableProperty

٣- قم بتعيين خاصية BindingContext الخاصة ب Grid

يعد تعيين BindingContext على Grid هو الخيار الأكثر أمانًا، ليست هناك احتمالية استخدام BindingContext مختلف، كمصدر للربط.

٤ تمرين - استبدال التعليمات بروابط .NET MAUI bindings

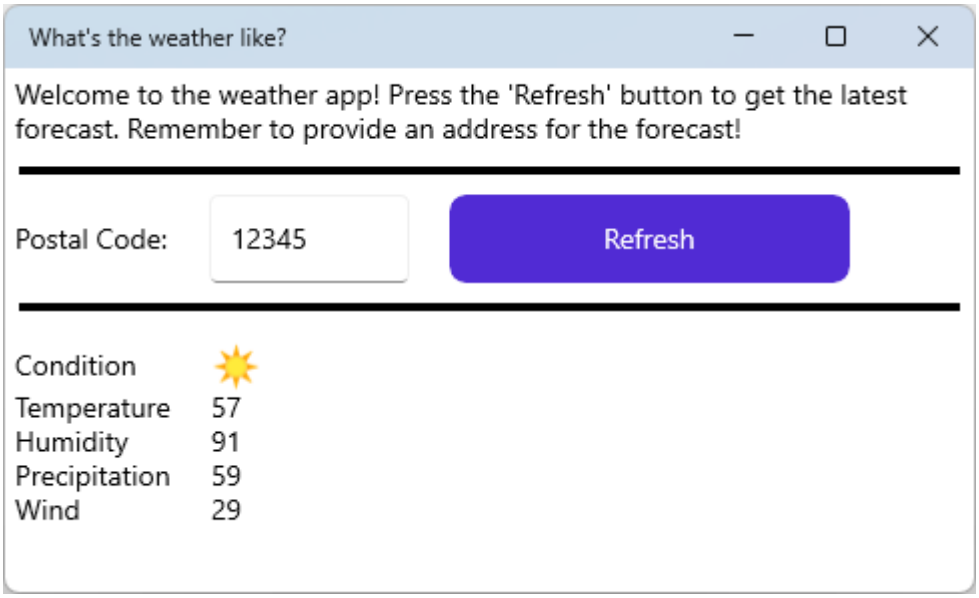
في هذا التمرين، ستقوم بتحويل تطبيق يستخدم الأحداث والبرمجة الخلفية events and code-behind إلى تطبيق يستخدم غالباً ربط البيانات data binding تطبيق التمرين هو تطبيق للتنبؤ بالطقس يعرض الطقس لهذا اليوم.

افتح the starter solution

١- نسخ أو تنزيل exercise repo من GitHub

٢- افتح WeatherClient.sln solution من مجلد start folder باستخدام Visual Studio أو المجلد في Visual Studio Code

٣- إنشاء المشروع وتشغيله Build and run للتأكد من أنه يعمل، على الشاشة المعروضة، سترى بعض تفاصيل الطقس الفارغة، اضغط على الزر **Refresh** button وسترى تحديث تفاصيل الطقس.



٤- للرجوع إليها، إليك ملخصاً للفئات والملفات classes and files التي ستعمل معها في هذا التمرين:

MainPage.xaml

تعريف واجهة المستخدم والمنطق للصفحة الأولية UI and logic for the initial page يعرّف ملف XAML واجهة المستخدم باستخدام العلامات markup

MainPage.xaml.cs

يعرف واجهة المستخدم والمنطق للصفحة الأولية UI and logic for the initial page ملف التعليمات الخلفي المرتبط الذي يحتوي على التعليمات البرمجية المتعلقة بواجهة المستخدم المعرفة بواسطة MainPage.xaml

Services\WeatherService.cs

تحاكي هذه الفئة خدمة إعداد تقارير الطقس، تحوي أسلوب واحد يسمى `GetWeather` والتي تقوم بإرجاع `WeatherData type`

Models\WeatherData.cs

يحتوي على بيانات الطقس. وهو عبارة عن نوع بسيط من السجلات يوفر درجة الحرارة وهطول الأمطار والرطوبة والرياح والحالة الجوية لهذا اليوم.

Models\WeatherType.cs

تعداد حالة الطقس، مشمس أو غائم.

تعيين سياق الربط Set the binding context

ستحتاج إلى تحرير التعليمات الخلفية لمعالج حدث click زر Refresh button تحصل التعليمات حالياً على بيانات الطقس وتحديث عناصر التحكم مباشرة، بدلاً من ذلك، احصل على بيانات الطقس وقم بتعيينها كسياق ربط للصفحة the binding context for the page

١- افتح ملف التعليمات البرمجية MainPage.xaml.cs

٢- راجع الأسلوب `btnRefresh_Clicked` method ينفذ هذا الأسلوب الخطوات التالية:

- يقوم بتعطيل الزر Disables the button وتمكين "enables the "busy spinner
- الحصول على توقعات الطقس weather forecast من خدمة الطقس weather service
- تحديث عناصر التحكم على الصفحة controls on the page بمعلومات الطقس weather information

- تفعيل الزر Enables the button وتعطيل "busy" disables the spinner

٣- قم بإزالة التعليمات البرمجية التي تحدث عناصر التحكم بالبيانات. يجب أن تبدو التعليمات للحدث مثل المقطع التالي:

```
private async void btnRefresh_Clicked(object
sender, EventArgs e)
{
   btnRefresh.IsEnabled = false;
   actIsBusy.IsRunning = true;

Models.WeatherData weatherData = await
Services.WeatherServer.GetWeather(txtPostalCode.Text);

   btnRefresh.IsEnabled = true;
   actIsBusy.IsRunning = false;
}
```

٤- بدلاً من تعيين نتيجة أسلوب GetWeather method الخاص بالخدمة إلى متغير variable قم بتعيينها إلى `BindingContext` الخاصة بالصفحة:

```
private async void btnRefresh_Clicked(object
sender, EventArgs e)
{
   btnRefresh.IsEnabled = false;
   actIsBusy.IsRunning = true;

BindingContext = await
Services.WeatherServer.GetWeather(txtPostalCode.Text);

   btnRefresh.IsEnabled = true;
   actIsBusy.IsRunning = false;
}
```

٥- قم بتشغيل المشروع. لاحظ أنه عند الضغط على زر Refresh وإرجاع خدمة الطقس weather service للبيانات، لا يتم تحديث أي من عناصر التحكم بتوقعات الطقس the weather forecast سوف تقوم بإصلاح هذا الخطأ في القسم التالي.

إنشاء روابط bindings في XAML

الآن بعد أن قامت التعليمات الخلفية بتعيين سياق الربط للصفحة sets the binding context يمكنك إضافة الروابط إلى عناصر التحكم لاستخدام البيانات الموجودة في السياق the data on the context

١- افتح الملف MainPage.xaml

٢- ابحث عن Grid الداخلية التي تحتوي على كافة عناصر Label

```
<Grid Grid.Row="2" RowDefinitions="Auto, Auto, Auto,
Auto, Auto" ColumnDefinitions="Auto, Auto"
Margin="0,5,0,0">
    <Label Grid.Row="0" Grid.Column="0"
Text="Condition" VerticalOptions="Center" />
    <Image x:Name="imgCondition" Grid.Row="0"
Grid.Column="1" HeightRequest="25" WidthRequest="25"
Source="question.png" HorizontalOptions="Start" />
    <Label Grid.Row="1" Grid.Column="0"
Text="Temperature" Margin="0,0,20,0" />
    <Label x:Name="lblTemperature" Grid.Row="1"
Grid.Column="1" Text="0" />
    <Label Grid.Row="2" Grid.Column="0"
Text="Humidity" Margin="0,0,20,0" />
    <Label x:Name="lblHumidity" Grid.Row="2"
Grid.Column="1" Text="0" />
    <Label Grid.Row="3" Grid.Column="0"
Text="Precipitation" Margin="0,0,20,0" />
    <Label x:Name="lblPrecipitation" Grid.Row="3"
Grid.Column="1" Text="0" />
    <Label Grid.Row="4" Grid.Column="0" Text="Wind"
Margin="0,0,20,0" />
    <Label x:Name="lblWind" Grid.Row="4"
Grid.Column="1" Text="0" />
</Grid>
```

٣- أضف روابط bindings إلى كل عنصر تحكم مُسمى Label هناك أربعة عناصر تحكم.

يجب تغيير قيمة خاصية `Label.Text` إلى بناء الجملة `{Binding PROPERTY_NAME}` حيث `PROPERTY_NAME` هي خاصية من نوع `Models.WeatherData` المحدد في Models\WeatherData.cs تذكر أن هذا النوع هو نوع البيانات الذي تم إرجاعه بواسطة خدمة الطقس weather service

على سبيل المثال، يجب أن تكون خاصية `Text` لعنصر Label المسمى `lblWind` (Label الأخير في Grid) تبدو مثل التعليمات التالية:

```
<Label x:Name="lblWind" Grid.Row="4"
Grid.Column="1" Text="{Binding Wind}" />
```

٤- ضمن `<Grid>` عناصر التحكم التي تسرد جميع تفاصيل الطقس weather details قم بإزالة جميع السمات attributes `x:Name="..."`

الأسماء names غير مطلوبة الآن، حيث لم تعد عناصر التحكم مذكورة في التعليمات البرمجية الخلفية the code-behind

٥- تحقق من أن روابط XAML bindings تطابق مقطع التعليمات التالي:

```
<Grid Grid.Row="2" RowDefinitions="Auto, Auto,
Auto, Auto, Auto" ColumnDefinitions="Auto, Auto"
Margin="0,5,0,0">
    <Label Grid.Row="0" Grid.Column="0"
Text="Condition" VerticalOptions="Center" />
    <Image Grid.Row="0" Grid.Column="1"
HeightRequest="25" WidthRequest="25"
Source="question.png" HorizontalOptions="Start" />
    <Label Grid.Row="1" Grid.Column="0"
Text="Temperature" Margin="0,0,20,0" />
    <Label Grid.Row="1" Grid.Column="1"
Text="{Binding Temperature}" />
    <Label Grid.Row="2" Grid.Column="0"
Text="Humidity" Margin="0,0,20,0" />
    <Label Grid.Row="2" Grid.Column="1"
Text="{Binding Humidity}" />
    <Label Grid.Row="3" Grid.Column="0"
Text="Precipitation" Margin="0,0,20,0" />
    <Label Grid.Row="3" Grid.Column="1"
Text="{Binding Precipitation}" />
    <Label Grid.Row="4" Grid.Column="0"
Text="Wind" Margin="0,0,20,0" />
    <Label Grid.Row="4" Grid.Column="1"
Text="{Binding Wind}" />
</Grid>
```

٦- قم بتشغيل التطبيق واضغط على الزر Refresh button يعمل التطبيق تقريباً مثل الأصلي.

لاحظ أن الأيقونة icon التي تمثل الشرط **Condition** لا يتم تحديثها من علامة الاستفهام إلى أيقونة الشمس أو السحب، لماذا لا تتغير الأيقونة؟ لأن الأيقونة هي مورد صورة image resource تم اختياره في التعليمات البرمجية استنادًا إلى قيمة التعداد `WeatherData.Condition` enumeration value لا يمكن تغيير قيمة التعداد enumeration value إلى مورد صورة image resource دون بذل جهد إضافي، يتم إصلاح هذا في التمرين التالي بعد معرفة المزيد حول الروابط bindings

٥ ربط البيانات Data binding مع أنواع غير متطابقة mismatched types

في بعض الأحيان لا تتطابق البيانات التي تستخدمها مع نوع بيانات خاصية عنصر التحكم التي تعرض البيانات، على سبيل المثال، قد يكون لديك قيمة decimal type مخزنة، تريد عرضها على عنصر Label منسقة كعملة currency ومن الأمثلة الأكثر تعقيدًا تطبيق الطقس المقدم في الوحدة، من المفترض أن يتم عرض صورة بناءً على قيمة تعداد "مشمس" أو "غائم" لتوقعات الطقس the weather forecast's Sunny or Cloudy enumeration value ولكن لا يمكنك ربط قيمة التعداد source's enumeration value الخاصة بالمصدر بخاصية صورة الهدف target's image يبحث هذا الدرس في الطرق التي يمكنك من خلالها تحويل البيانات convert data

تنسيق السلسلة النصية String formatting

من بين حالات عدم التطابق الشائعة بين الأنواع، نوع مُضمن تريد عرضه كسلسلة مُنسَّقة، كما هو الحال عندما تريد عرض أجزاء من قيمة DateTime أو تريد تنسيق نوع عشري كعملة.

لنفترض أنك تريد عرض المبلغ المستحق على فاتورة ولديك هذه الخاصية على كائن البيانات الخاص بك:

```
public decimal BillAmount { get; set; }
```

ينتهي المبلغ المستحق إلى **22.0304** يمكنك استخدام عنصري تحكم Label لعرض بعض النص والمبلغ بالدولار، كما هو موضح في التعليمات التالية:

```
<HorizontalStackLayout>
  <Label Text="You owe" Margin="0,0,5,0" />
  <Label Text="{Binding BillAmount}" />
  <Label Text="to the bank" Margin="5,0,0,0" />
</HorizontalStackLayout>
```

يؤدي هذا إلى إخراج سلسلة string إلى واجهة المستخدم تبدو مثل `You owe 22.0304 to the bank` ولكنها تفتقد رمز العملة، وقد يكون هناك عدد كبير جدًا أو عدد قليل جدًا من المنازل العشرية، بناءً على العملة المحلية. عادةً ما تقوم بمعالجة القيمة كسلسلة string مع محدد تنسيق "C" (أو العملة) "C" (or currency) format في التعليمات البرمجية، مثل ذلك:

```
string formattedBillAmount =
string.Format("{0:C}", BillAmount);
```

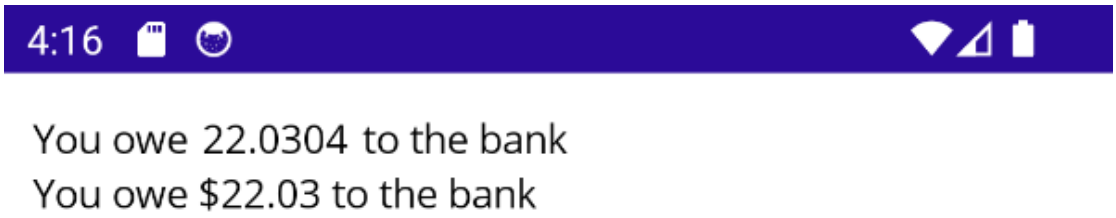
ولكن لاستخدام التنسيق في ربط البيانات formatting in data binding يجب أن يوفر لك كائن البيانات data object هذه السلسلة المنسقة كخاصية مختلفة، أو اعتراضها بطريقة ما وتنسيقها بنفسك، لحسن الحظ، توفر روابط .NET MAUI bindings طريقة لتنسيق السلاسل format strings باستخدام خاصية الربط StringFormat binding تتبع سلسلة التنسيق نفس القواعد المُتبعة في أسلوب String.Format method احرص على إحاطة سلسلة التنسيق بعلامات اقتباس مفردة single quotes بحيث لا يرتبك محلل XAML بسبب الأقواس المتعرجة، معلمة تنسيق السلسلة 0 String format parameter هي قيمة الخاصية التي تتم معالجتها بواسطة الربط.

```
<Label Text="{Binding BillAmount,
StringFormat='You owe {0:C} to the bank'}" />
```

خذ في الاعتبار ملف XAML التالي، الذي يعرض BillAmount باستخدام كلتا الطريقتين:

```
<VerticalStackLayout Padding="10">
  <HorizontalStackLayout>
     <Label Text="You owe" Margin="0,0,5,0" />
     <Label Text="{Binding BillAmount}" />
     <Label Text="to the bank" Margin="5,0,0,0" />
  </HorizontalStackLayout>
  <HorizontalStackLayout>
     <Label Text="{Binding BillAmount,
StringFormat='You owe {0:C} to the bank'}" />
  </HorizontalStackLayout>
</VerticalStackLayout>
```

توضح الصورة التالية ما ينتجه إخراج XAML على الشاشة:

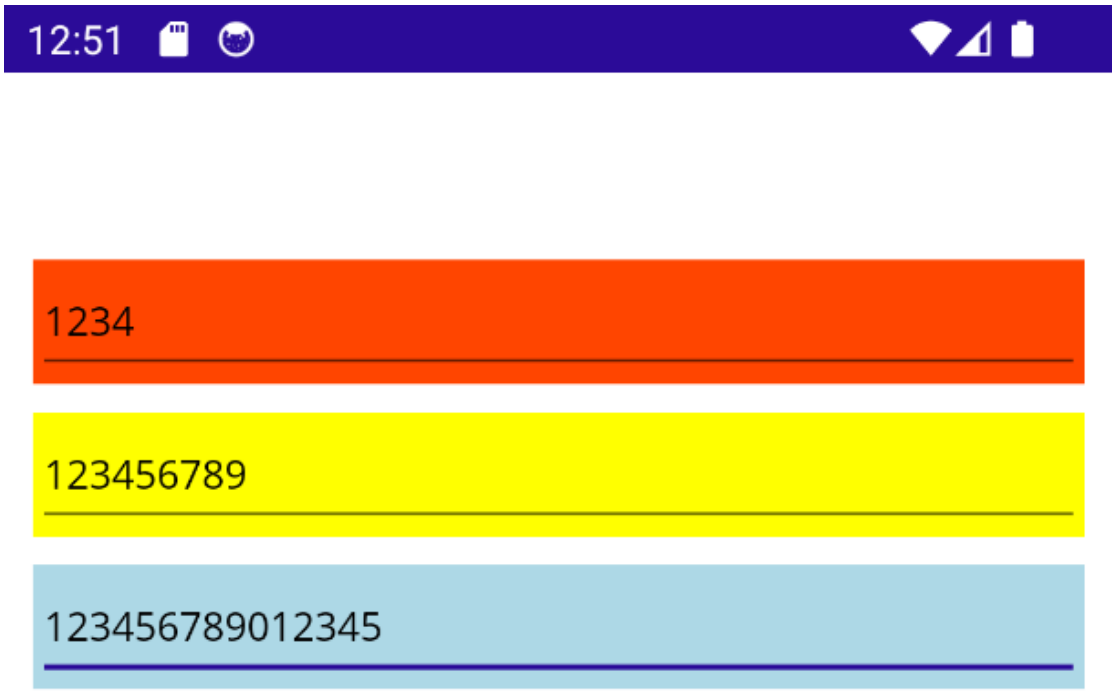


يعد XAML الذي يستخدم خاصية StringFormat binding أبسط من XAML الأخرى، ويمكنك الوصول إلى نظام تنسيق السلسلة القوي .NET.

تحويل نوع مُخصّص Custom type conversion

تعتبر خاصية الربط StringFormat binding ملائمة عند عرض قيمة كسلسلة string ولكن ليس عندما تريد تحويل شيء مثل لون أو صورة من نوع آخر Color or Image from another type في هذه الحالات، تحتاج إلى كتابة تعليمة تحويل مخصص.

لنفترض أنك تطالب المستخدم باختيار كلمة مرور، وتريد استخدام لون في واجهة المستخدم للإشارة إلى قوة كلمة المرور، هناك ثلاثة مستويات من القوة: ضعيفة، جيدة، قوية. تعتمد القوة على عدد الأحرف الموجودة في كلمة المرور، لتقديم ملاحظات فورية للمستخدم حول قوة كلمة المرور الخاصة به، تريد تغيير خلفية عنصر Entry الذي يحتوي على كلمة المرور استنادًا إلى القوة. توضح الصورة التالية هذه المستويات الثلاثة من القوة: ضعيفة وجيدة وقوية weak, good, strong



يتم تعيين هذه المستويات إلى the `Strength` enumeration

```
private enum Strength
{
    Weak,
    Good,
    Strong
```

```
}
```

يتم تعيين كائن بيانات كـ `BindingContext` للصفحة، والذي يحتوي على قوة كلمة المرور باستخدام الخاصية `PasswordStrength` عندما يكتب المستخدم كلمة مرور، تتغير الخاصية `PasswordStrength` وفقًا لطول كلمة المرور. ولأن كائن البيانات data object يحتوي على الخاصية PasswordStrength فإنك تقوم بربط هذه الخاصية بلون الخلفية `BackgroundColor` لعنصر التحكم Entry

```
<Entry BackgroundColor="{Binding PasswordStrength}
... />
```

توجد مشكلة هنا رغم ذلك فـ PasswordStrength من النوع `Strength` بينما لون الخلفية BackgroundColor هو `Color` هذه الأنواع غير متوافقة مع بعضها البعض. يوفر .NET MAUI طريقة لإصلاح مشكلات عدم تطابق النوع type-mismatch هذه، وهي خاصية the binding's `Converter`

يقوم محول الربط binding converter بما يشير إليه اسمه تمامًا، أي التحويل بين مصدر الربط والهدف binding source and target يتم تنفيذ المحولات من خلال واجهة interface `IValueConverter`

تنفيذ Implement IValueConverter

يمكنك إنشاء منطق التحويل الخاص بك في فئة class تنفذ الواجهة interface `IValueConverter` وعادةً ما تنتهي أسماء هذه الفئات باسم `Converter` لتوضيح الغرض منه.

تحدد الواجهة `IValueConverter` أسلوبين two methods

- `Convert` التحويل من خاصية مصدر الربط the binding source's property إلى خاصية واجهة مستخدم هدف الربط the binding target's UI property
- `ConvertBack` التحويل من خاصية واجهة مستخدم هدف الربط the binding target's UI مرة أخرى إلى خاصية مصدر الربط the binding source's property

نادرًا ما يتم استخدام هذا الأسلوب method ولا يتم استخدامها في هذا السيناريو. تطرح معظم المحولات استثناء `NotSupportedException` للإشارة إلى أن هذا التحويل غير مدعوم.

إليك عقد الواجهة the interface's contract

```
public interface IValueConverter
{
 object? Convert(object? value, Type targetType,
object? parameter, CultureInfo culture);
 object? ConvertBack(object? value, Type targetType,
object? parameter, CultureInfo culture);
}
```

تذكر أن السيناريو الذي نعمل عليه هو ربط خاصية Entry.BackgroundColor بخاصية PasswordStrength الخاصة بكائن البيانات data object وهي عبارة عن تعداد Strength enumeration

```
<Entry BackgroundColor="{Binding PasswordStrength}
... />
```

يجب تحويل Strength enumeration إلى لون Color لذا يحتاج المحول إلى تقييم قيمة Strength value المقدمة وإرجاع لون مختلف Color توضح التعليمات البرمجية التالية هذا التحويل:

```
namespace MyProject.Converters;

class StrengthToColorConverter : IValueConverter
{
  public object? Convert(object? value, Type
targetType, object? parameter, CultureInfo culture)
   {
       return (Strength)value! switch
       {
           Strength.Weak => Colors.OrangeRed,
           Strength.Good => Colors.Yellow,
           Strength.Strong => Colors.LightBlue,
           _ => Colors.LightBlue
       };
   }

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

```
}
```

دعونا نحلل هذه التعليمة البرمجية:

- يحتوي أسلوب `Convert` method على أربع معلمات parameters يمكنك عمومًا تجاهل المعلمتين الأخيرتين ما لم يكن لديك سبب محدد لاستخدامهما.
- تحتوي معلمة `value` parameter على القيمة الواردة incoming value في هذا المثال، إنها قيمة Strength enumeration
- يتم تجاهل معلمة `targetType` parameter ولكن يمكنك استخدام هذه المعلمة للتحقق من أن نوع الخاصية التي يستخدمها المحول هو `Color` تم حذف هذه المعلمة في هذا المثال من أجل التبسيط.
- يتم استخدام تعبير التبديل switch expression لإرجاع لون color مختلف استنادًا إلى value `Strength`

استخدام المحول the converter في XAML

مع إنشاء فئة المحول converter class تحتاج إلى إنشاء مثيل له والإشارة إليه في الربط، الطريقة القياسية لإنشاء مثيل للمحول instantiate the converter موجودة في قاموس موارد العنصر الجذر the root element's resource dictionary

أولاً، قم بتعيين `XML namespace` إلى مساحة اسم `.NET namespace` التي تحتوي على المحول converter في مثال التعليمات البرمجية التالي، يتم تعيين مساحة اسم `XML namespace` إلى مساحة اسم `.NET namespace` `MyProject.Converters`

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
   xmlns:cvt="clr-namespace:MyProject.Converters"
     ...
```

بعد ذلك، قم بإنشاء مثيل في `ContentPage.Resources`

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:cvt="clr-namespace:MyProject.Converters"
   x:Class="MyProject.PasswordExample">
   <ContentPage.Resources>
      <cvt:StrengthToColorConverter
x:Key="StrengthToColorConverter" />
```

```
  </ContentPage.Resources>
```

الآن يوجد مثيل للمحول instance of the converter في قاموس الموارد resource dictionary بالمفتاح key `StrengthToColorConverter` والذي يكون له نفس اسم النوع، هذه طريقة نموذجية لتسمية المحولات converters حيث يكون لديك عادةً محول واحد، تعيد استخدامه في جميع أنحاء XAML إذا كنت، لسبب ما، تحتاج إلى مثيلات متعددة للمحول، فيجب أن تكون المفاتيح مختلفة فيما بينها.

أخيرًا، قم بالإشارة إلى المحول في الرابط، نظراً لأن المحول موجود في قاموس الموارد، يمكنك استخدام ملحق العلامات markup `{StaticResource}` extension للإشارة إليه. يتم تعيين المحول إلى خاصية binding `Converter`

```
<Entry BackgroundColor="{Binding
PasswordStrength, Converter={StaticResource
StrengthToColorConverter}}" ... />
```

اختبار بسيط:

١- لعرض وتنسيق قيمة كسلسلة value as a string في ربط البيانات data binding ما هو أفضل نهج؟

٢- ما هو النوع type المستخدم في تحويل ربط البيانات data binding conversion

حل الاختبار:

١- استخدم خاصية `StringFormat binding`

يمكنك تنسيق البيانات باستخدام الربط، يسهل هذا الخيار تغيير تنسيق السلسلة string format بناءً على كيفية عرض واجهة المستخدم للبيانات

٢- `IValueConverter`

يتم استخدام هذه الواجهة بواسطة نظام ربط البيانات the data binding system ويمكنك تنفيذها لإنشاء فئات تحويل البيانات data conversion classes الخاصة بك.

٦ تمرين - إنشاء محول .NET MAUI converter

في هذا التمرين، يمكنك إضافة محول إلى تطبيق الطقس الذي تم إنشاؤه في التمرين السابق، المحول الأول يحول قيمة تعداد enumeration value إلى مورد صورة image resource المحول الثاني يحول درجة الحرارة the temperature من فهرنهايت Fahrenheit إلى درجة مئوية Celsius

التحويل إلى صورة Convert to an image

السياق الحالي لربط صفحة تطبيق الطقس هو كائن بيانات يحتوي على خصائص تصف توقعات الطقس. إحدى هذه الخصائص هي حالة السماء the sky condition وهي عبارة عن enumeration عند عرض معلومات الطقس، يجب أن يعرض التطبيق أيقونة لمساعدة المستخدم على تصور حالة السماء the sky condition لعرض هذه الأيقونات، يجب تحويل enumeration إلى مورد صورة image resource

١- افتح مشروع Weather Sample من التمرين السابق في Visual Studio أو Visual Studio Code

٢- إضافة مجلد إلى المشروع المسمى Converters

٣- أضف فئة جديدة new class إلى مجلد المحولات Converters folder المسمى WeatherConditionToImageConverter.cs

٤- افتح WeatherConditionToImageConverter.cs في محرر التعليمات البرمجية، واستبدل جميع التعليمات بالتعليمات التالية:

```
using System.Globalization;
using WeatherClient.Models;

namespace WeatherClient.Converters;

internal class WeatherConditionToImageConverter :
IValueConverter
{
   public object? Convert(object? value, Type
targetType, object? parameter, CultureInfo culture)
    {
      WeatherType weatherCondition =
(WeatherType)value!;
```

```
        return weatherCondition switch
        {
            Models.WeatherType.Sunny =>
ImageSource.FromFile("sunny.png"),
            Models.WeatherType.Cloudy =>
ImageSource.FromFile("cloud.png"),
            _ =>
ImageSource.FromFile("question.png")
        };
    }

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

`WeatherConditionToImageConverter` تعرف هذه التعليمات المحول في مساحة الاسم `WeatherClient.Converters` namespace يتوقع هذا المحول قيمة `WeatherType` enumeration ويرجع مورد صورة بناءً على هذه القيمة.

٥- أفتح ملف MainPage.xaml

٦- في العنصر الجذر، the root element أضف مساحة اسم جديدة new XML namespace باسم `cvt` وقم بتعيينها إلى .NET namespace `WeatherClient.Converters`

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
   xmlns:cvt="clr-namespace:WeatherClient.Converters"
   x:Class="WeatherClient.MainPage">
```

٧- أضف مثيلًا لمحول `WeatherConditionToImageConverter` إلى موارد الصفحة the page's resources باستخدام key of `WeatherConditionToImageConverter`

```
<ContentPage ...
```

```
  <ContentPage.Resources>
    <cvt:WeatherConditionToImageConverter x:Key=
"WeatherConditionToImageConverter" />
 </ContentPage.Resources>
```

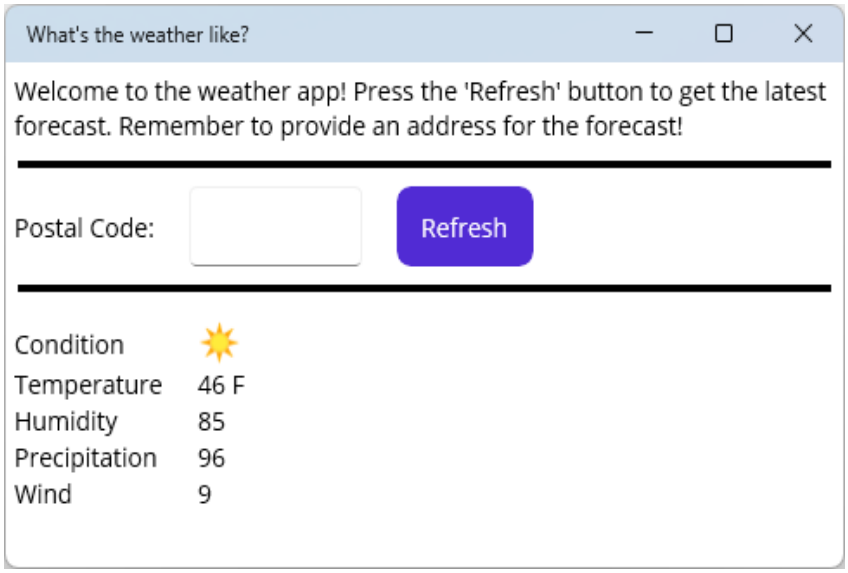
٨- أعثر على العنصر `<Image>` في `Grid.Row="0"`

٩- غير الخاصية `Source="question.png"` إلى الربط binding التالي:

```
Source="{Binding Condition, Converter={StaticResource
WeatherConditionToImageConverter}}"
```

١٠- شغل البرنامج.

لاحظ أنه عند الضغط على Refresh button يتغير حقل الحالة Condition field إلى أيقونة:



٧ الملخص

يستخدم تطبيق الطقس الذي تقوم بصيانته خدمة الطقس، التي أعلنت مؤخرًا عن توقفها، كان عليك البدء في التحقق من خدمات طقس أخرى، ولأنك تعلم أن تغيير الخدمات يعني أن بياناتك قد تتغير، فقد أردت التبديل من استخدام التعليمات البرمجية الخلفية code-behind لتحديث واجهة المستخدم إلى ربط البيانات data binding وبهذه الطريقة، عندما تقرر خدمة الطقس التي ستستخدمها، فلن تضطر إلى القلق بشأن تأثر واجهة المستخدم.

قلل ربط البيانات الذي قمت به في التمارين من حجم التعليمات البرمجية المطلوبة لتحديث واجهة المستخدم عند تغير البيانات. فبدلاً من استخدام معالج أحداث الزر button's event handler للتفاعل مع عناصر التحكم في واجهة المستخدم لتقديم البيانات، انتقلت إلى ربط البيانات data binding وتم تعيين كائن بيانات خدمة الطقس The weather service's data object كسياق ربط للصفحة the binding context for the page وربط عناصر التحكم في الصفحة بخصائص كائن البيانات هذا، وتمت مزامنة واجهة المستخدم تلقائيًا مع البيانات، بغض النظر عن كيفية تحديث خدمة الطقس. تمكنت أيضًا من تقليل التعليمات البرمجية الخلفية للتحويل من فهرنهايت إلى مئوية باستخدام ربط البيانات data binding مع محول converter

تخيل أن التعليمات البرمجية الخلفية قد تأثر دون ربط البيانات، إذا قمت بإعادة تسمية عنصر تحكم، أو تغيير عنصر تحكم واحد إلى نوع مختلف، أو حتى حذف عنصر تحكم، فلن يتم تجميع التعليمات البرمجية الخلفية برمجياً، إذا توقفت خدمة الطقس عن توفير حقل معين من البيانات، مثل الرطوبة the humidity فستتعطل التعليمات البرمجية عند محاولة تقديمه في واجهة المستخدم، وسوف يتوقف التطبيق عن العمل فجأة ولن يعرف المستخدمون ما الذي يحدث.

يقوم ربط البيانات بمزامنة البيانات تلقائيًا مع واجهة المستخدم. وبمجرد تغير بيانات الطقس، يتغير أيضًا أي شيء مرتبط بها، عندما لا يتطابق نوع خاصية واجهة المستخدم ونوع كائن البيانات، يقوم المحول بتحويل البيانات المرتبطة بحيث يتم تقديمها بشكل صحيح بواسطة واجهة المستخدم، وهذا يقلل من التعليمات البرمجية الخلفية المطلوبة للحفاظ على واجهة المستخدم.

من أين تأتي البيانات وكيف تم تشغيل البيانات ليس مصدر قلق لمعظم واجهة المستخدم الخاصة بك. إذا لم تعد الرطوبة the humidity متوفرة في كائن البيانات، فلن تظهر واجهة المستخدم المرتبطة بالبيانات أي شيء لتسمية الرطوبة بدلاً من التعطل، هذه تجربة أفضل بكثير للمستخدمين بدلاً من تطبيق متعطل.

الوحدة السابعة

استخدام خدمات ويب REST في تطبيقات .NET MAUI.

استخدام خدمة ويب REST web service باستخدام HttpClient وتنفيذ عمليات basic CRUD operations الأساسية. ستكتشف متى يكون جهازك متصلاً بالإنترنت لتوفير تجربة مستخدم جيدة والاستفادة من مجموعات الشبكات الأصلية the native networking stacks للحصول على أعلى أداء.

الأهداف التعليمية

خلال هذه الوحدة، سوف تتمكن مما يلي:

- اكتشف ما إذا كان جهازك متصلا بالإنترنت.
- استهلاك خدمة ويب REST web service باستخدام HttpClient
- استفيد من مجموعات الشبكات الأصلية أثناء استخدام HttpClient
- تكوين أمان شبكة العميل باستخدام ميزات الشبكات الأصلية لنظام التشغيل

محتويات الوحدة:

١- مقدمة

٢- الكشف عن اتصال الشبكة network connectivity

٣- استخدام خدمة REST service مع HttpClient

٤- استخدام ميزات الشبكة لنظام تشغيل محدد platform-specific network

٥- تمرين - استخدام خدمة REST service مع HttpClient

٦- الملخص

١ المقدمة

تستخدم العديد من التطبيقات الحديثة خدمات الويب REST web services لتوفير الوصول إلى البيانات أو ميزات أخرى مثل التخزين السحابي cloud storage ينطبق هذا بشكل خاص على تطبيقات الأجهزة المحمولة التي تعمل على الهواتف والأجهزة اللوحية، تعتمد معظم التطبيقات التي تتضمن الجوانب الاجتماعية والوسائط المشتركة على الاتصال بهذه الخدمات، بدون الاتصال، تكون العديد من التطبيقات محدودة في وظائفها، قد تقتصر على الميزات المخزنة مؤقتاً محلياً على الجهاز. يمكن للتطبيق المصمم جيدا الكشف بشفافية عما إذا كان اتصال الشبكة متوفراً وتعديل مجموعة ميزاته وفقاً لذلك.

تخيل أنك تعمل كمطور لشركة مرافق الطاقة power-utilities company أنت تقوم بإنشاء تطبيق يمكن لمهندسيك استخدامه عند زيارة مواقع العملاء لإجراء خدمة روتينية، أثناء زيارة الموقع، قد يحتاج المهندس إلى طلب أجزاء بديلة، يجب أن يسمح جزء من التطبيق للمهندس بالبحث بسرعة عن التفاصيل الخاصة بجزء ما، قامت شركتك مسبقاً بإنشاء خدمة ويب REST توفر معلومات حول المكونات والأجزاء الكهربائية، تعمل خدمة الويب هذه في Azure يجب أن يكون التطبيق قادراً على الاتصال بخدمة الويب هذه لاسترداد معلومات حول الأجزاء.

في هذه الوحدة، ستقوم بإنشاء تطبيق NET MAUI. الذي يستهلك البيانات من خدمة ويب REST ستبدأ بتحديد ما إذا كان جهازك متصلاً بالإنترنت، ومحاولة الاتصال بخدمة الويب فقط إذا كان لديك اتصال نشط، بعد ذلك، ستستخدم HttpClient لتنفيذ عمليات basic CRUD operations الأساسية على خدمة ويب REST مستضافة. وأخيراً، ستقوم بتكوين مجموعات الشبكات الأصلية the native networking stacks على كل جهاز لتنفيذ اتصالات آمنة بين التطبيق وخدمة الويب.

٢ الكشف عن اتصال الشبكة network connectivity

تستخدم الأجهزة المحمولة تقنيات Wi-Fi والهواتف المحمولة للاتصال بالإنترنت، تعني هذه التبعية أن المستخدمين قد يفقدون اتصالهم بالإنترنت في أثناء استخدام التطبيق، إذا لم تقم بإضافة تعليمات برمجية للحماية من هذه الإمكانية، فقد يتوقف تطبيقك عن الاستجابة، ويقدم للمستخدمين تجربة سيئة.

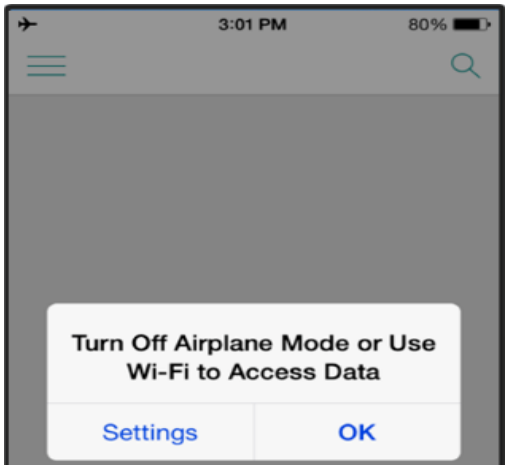
في هذا الدرس، يمكنك حماية التطبيق عن طريق الكشف عن الوقت الذي يفقد فيه المستخدمون اتصالهم بالإنترنت، هذه المعلومات تصبح هامة عندما نبدأ استخدام خدمات الويب REST

لماذا يتم الكشف عن اتصال الشبكة على تطبيقات الهاتف المحمول؟

يعد الكشف عما إذا كان لديك اتصال بالإنترنت على تطبيق جوال، أمراً مهماً لأن الأجهزة المحمولة قد تفقد اتصالها بشكل متكرر، قد يكون ذلك بسبب سوء التغطية من قبل موفر خدمة الشبكة، أو كونه في بيئة محدودة أو معزولة، مثل نفق أو وادي عميق أو جبل مرتفع، هناك أيضاً أنواع مختلفة من اتصال الشبكة، إذا كنت موجودًا في بيئة توفر اتصال WiFi فلديك عادة نطاق ترددي أعلى مما لو كنت تعتمد على الوصول إلى شبكة الجوال، قد تظل قادراً على الاتصال بالإنترنت، ولكن قد تكون بعض العمليات، مثل دفق محتوى الفيديو، أبطأ (ومكلفة) عبر ارتباط شبكة الجوال مقارنة باتصال WiFi

نظرًا إلى أن الأجهزة المحمولة تواجه هذه التحديات، عليك كتابة تعليمات برمجية للحماية منها، إذا لم تقم بذلك، وحاول تطبيقك تنفيذ عمليات تستخدم الإنترنت، فقد يتوقف التطبيق عن الاستجابة.

كما تريد توفير تجربة مستخدم جيدة عندما يتعذر على تطبيقك الاتصال بالإنترنت، إذا توقف تطبيقك عن العمل بسبب عدم وجود خدمة إنترنت، فربما يجعل المستخدمون في حيرة من أمرهم، أفضل شيء نفعله هو توفير المعلومات للمستخدمين، أخبرهم أنه ليس لديك اتصال بالإنترنت، وأن تطبيقك قد لا يعمل بشكل كامل بدونه، تعرض الصورة التالية مثالاً:



في هذا المثال، يُعلم مطور التطبيق المستخدم بأنه ليس لديه اتصال بالإنترنت، وأن عليه محاولة الاتصال بشبكة Wi-Fi

الكشف عن اتصال الشبكة

للتحقق من اتصال الشبكة في تطبيق .NET MAUI، استخدم الفئة Connectivity class تعرض هذه الفئة خاصية تسمى NetworkAccess وحدث يسمى ConnectivityChanged يمكنك استخدام هؤلاء الأعضاء للكشف عن التغييرات في الشبكة.

يمكنك الوصول إلى الخاصية NetworkAccess من خلال خاصية أخرى تسمى Current هذه هي الآلية التي تستخدمها Connectivity للوصول إلى التنفيذ الخاص بالنظام الأساسي.

ترجع خاصية NetworkAccess قيمة من قائمة تعداد the NetworkAccess enumeration تحتوي قائمة التعداد على خمس قيم: `ConstrainedInternet, Internet, Local, None, Unknown` إذا أعادت خاصية NetworkAccess قيمة NetworkAccess.None فهذا يعني أنك لا تملك اتصالاً بالإنترنت، ولا ينبغي لك تشغيل تعليمات الشبكة، هذه الآلية قابلة للنقل عبر الأنظمة الأساسية. تعرض التعليمات البرمجية التالية مثالاً.

```
if (Connectivity.Current.NetworkAccess ==
NetworkAccess.None)
{
    ...
}
```

يتيح لك حدث ConnectivityChanged أيضًا تحديد ما إذا كان الجهاز متصلاً بالإنترنت، يتم تشغيل حدث ConnectivityChanged تلقائيًا عند تغير حالة الشبكة، على سبيل المثال، إذا بدأت باتصال نشط بالشبكة ثم فقدته في النهاية، يتم تشغيل حدث ConnectivityChanged لإعلامك بالتغيير، أحد المعلمات parameters التي يتم تمريرها إلى معالج حدث ConnectivityChanged هو كائن ConnectivityChangedEventArgs object يحتوي هذا الكائن على خاصية تسمى IsConnected يُمكنك استخدام خاصية IsConnected لتحديد ما إذا كنت متصلاً بالإنترنت. إليك مثال:

```
Connectivity.Current.ConnectivityChanged +=
Connectivity_ConnectivityChanged;
...
void Connectivity_ConnectivityChanged(object
sender, ConnectivityChangedEventArgs  e)
```

```
{
    bool stillConnected = e.IsConnected;
}
```

يتيح لك حدث ConnectivityChanged كتابة تطبيقات يمكنها اكتشاف التغيير في حالة الشبكة وضبط الوظائف المتاحة بسلاسة وفقًا للبيئات المختلفة.

اختبار بسيط:

١- ما هو أفضل سبب للتحقق من اتصالك بالإنترنت قبل تشغيل التعليمات البرمجية للشبكة؟

٢- لنفترض أنك تكتب تطبيقاً يمكن للمستخدم تشغيله أثناء السفر في سيارة، إذا دخلت السيارة نفقاً، فقد يفقد الجهاز الاتصال بالإنترنت. أي عضو من فئة Connectivity class يجب استخدامه للكشف عن التغيير في اتصال الشبكة؟"

حل الاختبار:

١- تريد توفير تجربة مستخدم جيدة، إذا كان اتصال الشبكة محدودًا أو غير متوفر.

٢- The ConnectivityChanged event

٣ استخدام خدمة REST service مع HttpClient

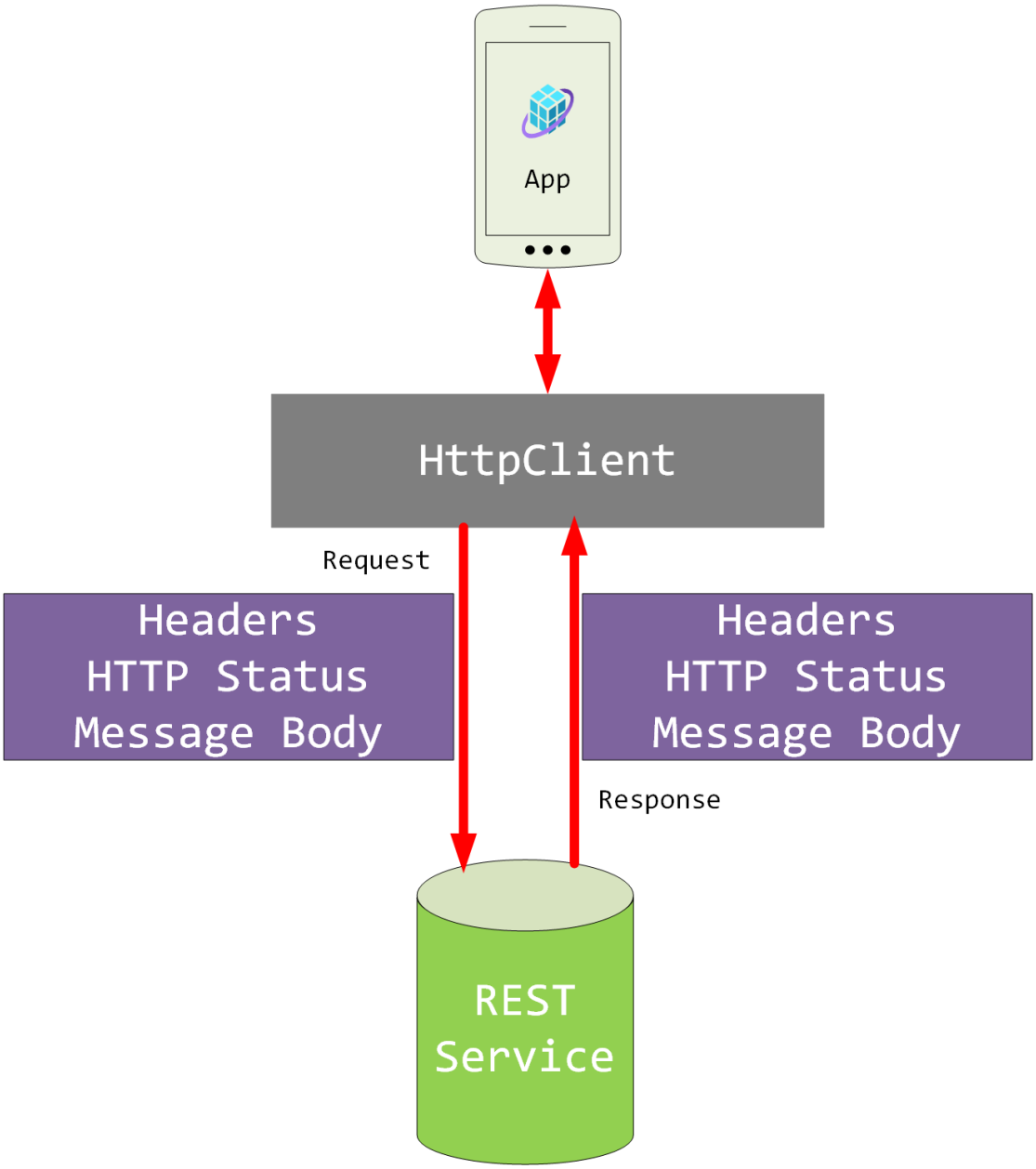
تنفذ العديد من خدمات الويب الحديثة بنية the REST architecture تمكن هذه البنية خدمة الويب من عرض العمليات والبيانات، من خلال سلسلة من نقاط النهاية المحددة جيدًا، تستخدم الطلبات التي ترسلها تطبيقات العميل إلى خدمة ويب REST لاسترداد البيانات أو تعديلها أو إنشائها أو حذفها مجموعة محددة مسبقاً من الأفعال، تستجيب خدمة ويب REST لهذه الطلبات بطريقة قياسية، يسهل هذا النهج إنشاء تطبيقات العميل.

تم إنشاء نموذج The REST model أعلى بروتوكول HTTP يمكن لتطبيق .NET MAUI إرسال طلبات إلى خدمة ويب REST web service باستخدام الفئة HttpClient class في هذا الدرس، ستتعرف على HttpClient كيفية استخدامها للتفاعل مع خدمة ويب REST

ما هي فئة HttpClient class

HttpClient هي فئة .NET class. يمكن للتطبيق استخدامها لإرسال طلبات HTTP وتلقي استجابات HTTP من خدمة ويب REST تحدد مجموعة من عناوين URI الموارد التي تعرضها خدمة الويب، يجمع عنوان URI بين عنوان خدمة الويب واسم المورد المتاح على هذا العنوان.

تستخدم الفئة HttpClient class واجهة برمجة التطبيقات القائمة على المهام لتحسين الأداء task-based API for performance وتمنحك حق الوصول إلى المعلومات الموجودة في رسائل الطلب مثل رؤوس HTTP ورمز الحالة status code بالإضافة إلى هيئات أو نصوص الرسائل التي تحتوي على البيانات الفعلية التي يتم إرسالها وتلقيها.



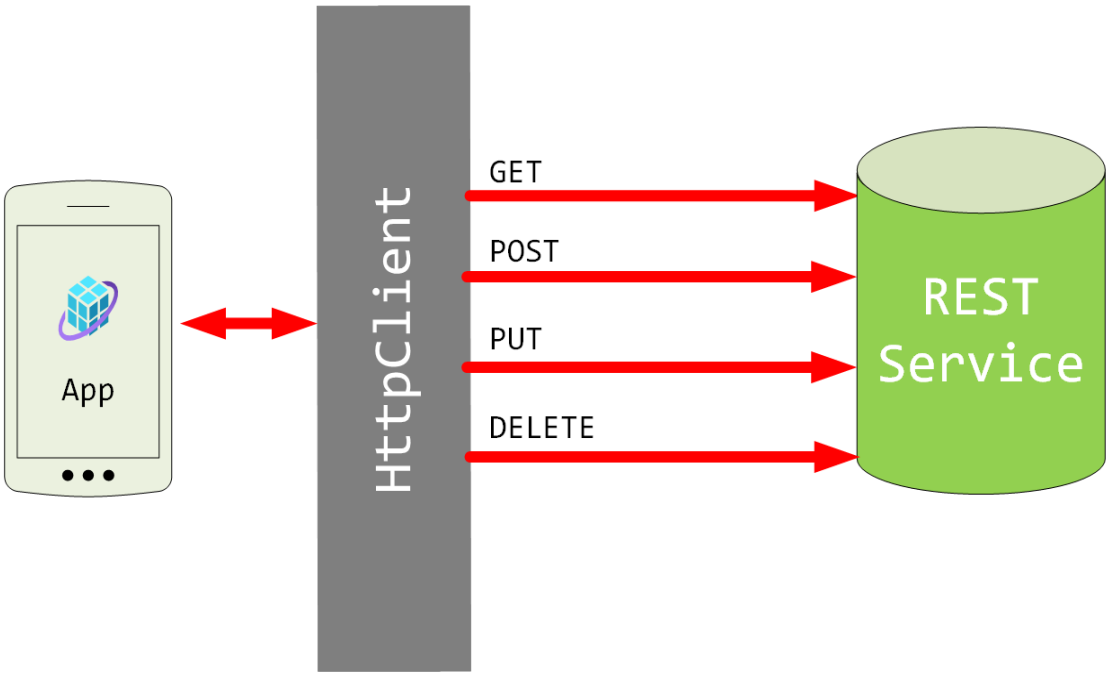
الفئة HttpClient class متوفرة في مساحة الاسم System.Net.Http namespace يمكن للتطبيق إنشاء كائن HttpClient object باستخدام الدالة الإنشائية الافتراضية the default constructor

```
using System.Net.Http;
...

var client = new HttpClient();
```

تنفيذ عمليات CRUD operations باستخدام كائن HttpClient object

تتيح خدمة ويب REST web service للعميل تنفيذ عمليات على البيانات من خلال مجموعة من أفعال set of HTTP verbs تتمثل مهمة فعل The HTTP verb's في الإشارة إلى الإجراء المطلوب تنفيذه على أحد الموارد resource هناك العديد من أفعال HTTP ولكن الأفعال الأربعة الأكثر شيوعًا هي POST, GET, PUT, DELETE يمكن للخدمة تنفيذ هذه الأفعال لتمكين تطبيق العميل من إدارة دورة حياة الكائنات the lifecycle of objects من خلال تنفيذ عمليات الإنشاء والقراءة والتحديث والحذف Create, Read, Update, Delete (CRUD) operations على النحو التالي:

- يُشير الفعل POST verb إلى رغبتك في إنشاء مورد جديد new resource
- يُشير الفعل GET verb إلى رغبتك في استرداد مورد ما retrieve a resource
- يُشير الفعل PUT verb إلى رغبتك في تحديث مورد ما update a resource
- يُشير الفعل DELETE verb إلى رغبتك في حذف مورد ما delete a resource



إنشاء مورد جديد new resource باستخدام HttpClient

لإنشاء مورد جديد باستخدام HttpClient يمكنك استخدام الأسلوب SendAsync method الذي يمرر له كائناً HttpRequestMessage object

يهدف HttpRequestMessage إلى نمذجة الطلب الذي يتم إرساله إلى خدمة الويب، يمكنك تحديد فعل HTTP verb وعنوان the web service's URL وقم بملء أي حمولة لإرسالها عبر الخاصية HttpRequestMessage.Content

```
HttpClient client = new HttpClient();

HttpRequestMessage message = new
HttpRequestMessage(HttpMethod.Post, url);
message.Content = JsonContent.Create<Part>(part);

HttpResponseMessage response = await
client.SendAsync(message);
```

يؤدي هذا الجزء من التعليمات البرمجية المهام التالية:

- يقوم بإنشاء مثيل instance of HttpClient يسمى client ويستخدمه لإرسال رسالة.
- يقوم بإنشاء مثيل لـ HttpRequestMessage يسمى message ويستخدمه لنمذجة الرسالة، تحتوي message على the HTTP verb and URL
- يقوم بتعيين خاصية Content لـ HttpRequestMessage باستخدام الدالة function JsonContent.Create تقوم هذه الدالة تلقائيًا بتحويل متغير الجزء part variable إلى JSON مناسب للإرسال إلى خدمة الويب.
- يقوم بإرسال الرسالة باستخدام كائن HttpClient object يتم إرجاع HttpResponseMessage التي تحتوي على معلومات مثل رمز الحالة the status code والمعلومات التي تم إرجاعها information returned بواسطة خدمة الويب.

قراءة مورد Read a resource باستخدام HttpClient

يمكنك قراءة مورد من خدمة ويب باستخدام نفس التقنية كما هو موضح سابقاً، باستثناء تهيئة HttpRequestMessage باستخدام HttpMethod.Get ومع ذلك، يحتوي HttpClient على اثنين من أساليب الراحة convenience methods التي توفر اختصارات.

لقراءة مورد باستخدام HTTPClient استخدم الأسلوب GetStringAsync method كما هو موضح في المثال التالي:

```
HttpClient client = new HttpClient();

string text = await client.GetStringAsync("https://...");
```

يأخذ الأسلوب GetStringAsync method عنوان URI الذي يشير إلى المورد resource ويعيد استجابة كسلسلة returns a response as a string استجابة السلسلة هي المورد الذي طلبه التطبيق، تنسيق بيانات الاستجابة The response-data format هو الافتراضي للخدمة المطلوبة، مثل JSON or XML يمكن للتطبيق أن يخبر خدمة الويب بأنه يحتاج إلى إرجاع البيانات بتنسيق معين عن طريق إضافة header MediaTypeWithQualityHeaderValue على سبيل المثال، إذا طلب التطبيق إرجاع البيانات بتنسيق JSON فيمكنه استخدام التعليمات البرمجية التالية:

```
HttpClient client = new HttpClient();

client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
```

في هذا المثال، يتم إرجاع النتيجة كسلسلة string وتحتوي فقط على نص رسالة الاستجابة the response message body للحصول على الاستجابة بأكملها بما في ذلك الرؤوس والنص ورمز الحالة headers, body, status code قم باستدعاء الأسلوب GetAsync method يتم إرجاع البيانات ككائن HttpResponseMessage object

تحديث مورد Update a resource باستخدام HttpClient

لتحديث مورد باستخدام HttpClient استخدم HttpRequestMessage الذي تمت تهيئته PUT verb التعليمات البرمجية التالية مشابهة لتلك المطلوبة لإنشاء مورد جديد:

```
HttpClient client = new HttpClient();
HttpRequestMessage message = new
HttpRequestMessage(HttpMethod.Put, url);

message.Content = JsonContent.Create<Part>(part);

HttpResponseMessage response = await
client.SendAsync(message);
```

ملاحظة:

الفرق الأساسي بين POST and PUT هو القدرة على تكرار نفس الطلب، إذا كررت نفس طلب PUT request عدة مرات، فسيتم تحديث نفس المورد بنفس البيانات، ويكون التأثير كما لو تم إرسال الطلب مرة واحدة فقط. إذا قمت بإصدار نفس طلب POST request عدة مرات، فستكون النتيجة هي قيام الخدمة the REST service بإنشاء نسخ متعددة من المورد the resource

حذف مورد Delete a resource باستخدام HttpClient

لحذف مورد باستخدام HttpClient قم باستدعاء SendAsync مع تهيئة HttpRequestMessage باستخدام فعل DELETE verb

```
HttpClient client = new HttpClient();
HttpRequestMessage message = new
HttpRequestMessage(HttpMethod.Delete, url);

HttpResponseMessage response = await
client.SendAsync(message);
```

وتحتوي الاستجابة على العناوين ورمز الحالة والعنصر المحذوف the headers, status code, deleted object

معالجة الاستجابات من طلب Handle responses from a request

ترجع جميع طلبات HTTP رسالة استجابة response message تعتمد البيانات الموجودة في الاستجابة على الفعل الذي أرسله التطبيق، على سبيل المثال، يحتوي نص الاستجابة the response body لطلب HTTP GET request على بيانات المورد resource المطلوب.

يرجع نص الاستجابة لطلب POST request نسخة من المورد الذي تم إنشاؤه، ولكن يجب أن يكون نص الاستجابة للطلب PUT فارغًا.

يجب عليك دائما التحقق من رمز الحالة status code والتعامل معه في رسالة الاستجابة، إذا كان رمز الحالة هذا في نطاق the 200 range (200, 201, 202,) وما إلى ذلك. فإن العملية تعتبر ناجحة، على الرغم من أنه قد تكون هناك حاجة إلى مزيد من المعلومات لاحقاً.

يشير رمز الحالة في النطاق the 300 range إلى أنه ربما تمت إعادة توجيه الطلب بواسطة خدمة الويب إلى عنوان مختلف، ربما نتيجة انتقال مورد إلى موقع مختلف.

يشير رمز الحالة في النطاق the 400 range إلى خطأ في العميل أو التطبيق، على سبيل المثال، رمز الحالة status code 403 يعني أن خدمة الويب تتطلب مصادقة

المستخدم، ولكن التطبيق لم يقم بذلك، يحدث رمز الحالة status code 403 عندما يحاول التطبيق الوصول إلى مورد غير موجود.

تشير رموز الحالة في النطاق the 500 range إلى وجود خطأ من جانب الخادم server-side error مثل عدم توفر الخدمة أو مشغولة جدًا لمعالجة الطلب.

يمكن للكائن HttpResponseMessage object الذي تم إرجاعه بواسطة طلب تم إرساله من خلال كائن HttpClient object أن يلخص قدرًا كبيرًا من التعقيد في التعامل مع رموز الحالة المختلفة، يوضح جزء التعليمات البرمجية هذا كيفية التحقق من أن رمز الحالة في رسالة استجابة يشير إلى النجاح، ومعالجة رموز الحالة التي تشير إلى نوع ما من الفشل.

```
static readonly HttpClient client = new HttpClient();

...
// Call asynchronous network methods in a
try/catch block to handle exceptions.
try
{
    //... Initiate the HttpRequest

    HttpResponseMessage response = await
client.SendAsync(msg);
    response.EnsureSuccessStatusCode(); // Check
that the status code is in the 200 range. Throw an
HttpRequestException if not

    string responseBody = await
response.Content.ReadAsStringAsync();

    ... // Handle the response
}
catch(HttpRequestException e)
{
    ... // Handle any status codes that indicate
an error.
        // The status code is available in the
field e.StatusCode
}
```

اختبار بسيط:

١- ما هو فعل HTTP verb الذي تستخدمه لإنشاء مورد resource جديد في خدمة ويب REST web service

٢- ما أسلوب فئة HttpResponseMessage class الذي يجب عليك استدعاؤها للتحقق من نجاح طلب HTTP request

حل الاختبار:

١- `POST`

٢- `EnsureSuccessStatusCode`

٤ استخدام ميزات الشبكة لنظام تشغيل محدد platform-specific network

توفر فئة HttpClient class تجريدًا للاتصال بالشبكة، التطبيق الذي يستخدم هذه الفئة مستقل عن مجموعة الشبكات الأصلية لنظام التشغيل the native platform networking stack تقوم قوالب The .NET MAUI templates بتعيين فئة HttpClient class إلى التعليمات البرمجية التي تستخدم مجموعة الشبكات الأصلية لكل نظام تشغيل، يتيح هذا للتطبيق الاستفادة من ميزات تكوين الشبكة الخاصة بنظام التشغيل وتحسينها، يعد هذا مهمًا بشكل خاص عندما تحتاج إلى تكوين تطبيق عميل للاتصال بشكل آمن بخدمة ويب REST web service

في هذا الدرس، ستتعلم كيفية تكوين تطبيق عميل HTTP client application لاستخدام إمكانيات حماية الشبكة التي يوفرها نظام التشغيل.

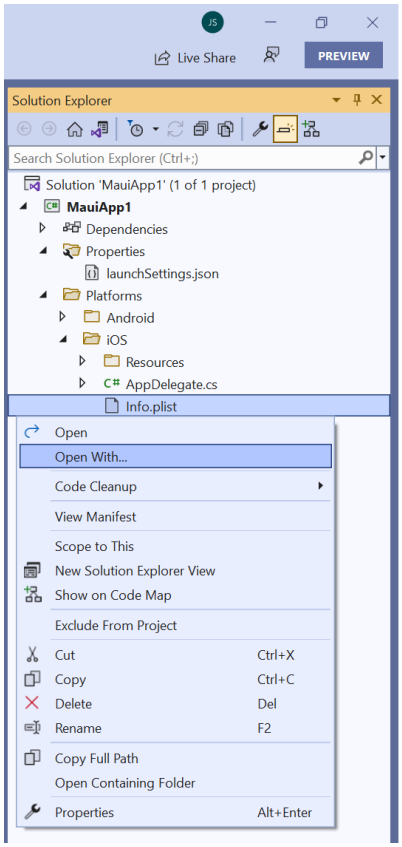
تكوين أمان App Transport Security على iOS

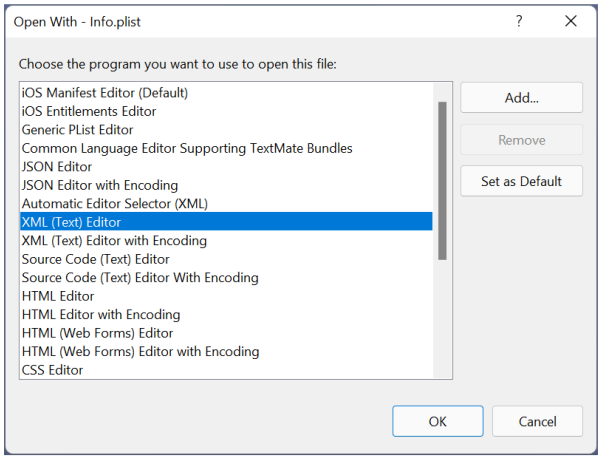
أمان نقل التطبيقات App Transport Security (ATS) هي ميزة في نظام التشغيل iOS وهذا يتطلب من كل اتصال شبكي يتم إجراؤه من خلال مجموعة شبكة the native HTTP network stack استخدام `TLS 1.2 or above` أو أعلى، لن تقوم خوارزميات التشفير الحديثة بالإفصاح عن المعلومات إذا تم اختراق أحد مفاتيح التشفير طويلة المدى the long-term keys

إذا لم يلتزم تطبيقك بهذه القواعد، فسيتم رفض وصوله إلى الشبكة، لإصلاح هذه المشكلة، لديك خياران؛ يمكنك تغيير نقطة النهاية الخاصة بك للالتزام بسياسة أمان نقل التطبيق the App Transport Security policy أو يمكنك إلغاء الاشتراك في أمان نقل التطبيق App Transport Security

لإلغاء الاشتراك في App Transport Security أضف مفتاحاً جديدًا new key يسمى `NSAppTransportSecurity` إلى ملف Info.plist ستجد ملف Info.plist في مجلد iOS في مجلد project's Platforms في مستكشف الحلول Solution Explorer هذا المفتاح key هو في الواقع قاموس.

أضف مفتاحًا آخر يسمى `NSExceptionDomains` إلى هذا القاموس dictionary يحتوي هذا المفتاح على مفتاح فرعي child لكل نقطة نهاية endpoints تريد استهدافها، يمكن أن يكون لكل نقطة نهاية تكوينها الخاص، مما يحدد الميزات التي يجب السماح بها أو عدم السماح بها، يمكنك إضافة هذا المفتاح إما باستخدام المحرر العام the generic plist editor في Visual Studio أو عن طريق فتحه كملف XML file





فيما يلي مثال على التكوين لنقطة نهاية واحدة configuration for one endpoint يتم عرضها بصيغة XML

```
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSExceptionDomains</key>
    <dict>
    <key>dotnet.microsoft.com</key>
    <dict>
      <key>NSExceptionMinimumTLSVersion</key>
      <string>TLSv1.0</string>
      <key>NSExceptionAllowsInsecureHTTPLoads</key>
      <true/>
    </dict>
  </dict>
</dict>
```

يضيف هذا المثال استثناءً إلى نقطة النهاية في dotnet.microsoft.com إذا كنت تقوم بتصحيح خدمة محلياً على جهاز التطوير الخاص بك، يمكنك إلغاء الاشتراك في App Transport Security بالنسبة لحركة مرور الشبكة المحلية باستخدام المفتاح local traffic with the `NSAllowsLocalNetworking` key كما يلي:

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsLocalNetworking</key>
    <true/>
</dict>
```

إذا لم تتمكن من تحديد جميع نقاط النهاية endpoints لديك، فقم بتعطيل أمان نقل التطبيق disable App Transport Security لجميع نقاط النهاية غير المحددة unspecified endpoints باستخدام المفتاح `NSAllowsArbitraryLoads` key

```
<key>NSAppTransportSecurity</key>
<dict>
   <key>NSAllowsArbitraryLoads</key>
   <true/>
</dict>
```

توجد خيارات أخرى يمكنك إضافتها لتحديد كيفية إلغاء الاشتراك بشكل أكثر تحديدًا، لا يشمل هذا الدرس الإرشادات الإضافية.

تكوين أمان شبكة اندرويد Configure Android Network security

مثل iOS يحتوي Android على نموذج أمان security model مشابه حول اتصال الشبكة، تم تقديم هذا النموذج model مع `Android 9 (API level 28)` يتم تعطيل حركة مرور النص الواضح Clear text (non-HTTPS) بشكل افتراضي عندما يستهدف تطبيقك Android 9 (API Level 28) أو أعلى، قد يؤثر هذا النهج على دورة التطوير إذا كان تطبيقك بحاجة إلى تنزيل صورة أو ملف على خادم لم يتم تكوينه لـ HTTPS أيضاً، قد تحاول فقط تصحيح تطبيقك محلياً ولا تريد تثبيت شهادات التطوير install development certificates ربما تكون لديك متطلبات عمل قوية مفادها أن كل حركة مرور الويب web traffic على جميع إصدارات Android تكون دائمًا عبر HTTPS تسمح لك ميزة تكوين أمان الشبكة The Network Security Configuration في Android ضبط أمان حركة مرور الشبكة بدقة tune network traffic security في التطبيق.

السماح بمسح حركة مرور النص Permit clear text traffic

للسماح بحركة مرور نصية واضحة، قم بإنشاء ملف XML جديد في المجلد Resources/xml باسم network_security_config.xml (قد تحتاج أيضًا إلى إنشاء مجلد xml) يوجد مجلد Resources في مجلد نظام Android في Solution Explorer داخل هذا الملف، أضف عنصر `network-security-config`

مع عنصر فرعي domain-config يتيح التكوين التالي حركة مرور نصية واضحة لنطاق معين وعنوان IP address

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="true">
    <domain
includeSubdomains="true">10.0.2.2</domain> <!--
Debug port -->
    <domain
includeSubdomains="true">microsoft.com</domain>
  </domain-config>
</network-security-config>
```

يمكنك تعزيز أمان تطبيقك من خلال تقييد حركة مرور النص الواضح restricting clear text traffic على جميع إصدارات Android بغض النظر عن إطار العمل المستهدف the target framework يمكنك القيام بذلك عن طريق تعيين خاصية cleartextTrafficPermitted لعنصر domain-config إلى false يحظر إعداد التكوين هذا جميع حركة مرور الشبكة غير HTTPS

لكي يتعرف التطبيق على ملف network_security_config.xml قم بتكوين خاصية networkSecurityConfig لعقدة application node في AndroidManifest.xml الموجودة في مجلد الخصائص Properties

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <application
android:networkSecurityConfig="@xml/network_security
_config" ...></application>
</manifest>
```

يمكنك تحديد خيارات إضافية إذا كنت بحاجة إلى أن تكون أكثر تحديدًا حول الطريقة التي تريد بها إلغاء الاشتراك في أمان النقل opt out of transport security

تصحيح أخطاء التطبيقات محلياً Debug apps locally

تتمثل إحدى المزايا المهمة لبناء تطبيقات الأجهزة المحمولة باستخدام Visual Studio في القدرة على تشغيل تطبيقات الجوال وتصحيحها باستخدام محاكي iOS أو محاكي Android يمكن أن تستهلك هذه التطبيقات خدمات الويب الأساسية ASP.NET Core web services التي تعمل محلياً ومكشوفة عبر HTTP

يمكن للتطبيقات التي تعمل في iOS Simulator الاتصال بخدمات ويب المحلية local HTTP web services باستخدام عنوان الجهاز machine's IP address أو عبر اسم مضيف المضيف المحلي the localhost hostname يجب أن يختار التطبيق إلغاء الاشتراك في ATS مع تحديد الحد الأدنى من `NSAllowsLocalNetworking` على سبيل المثال، بالنظر إلى خدمة ويب المحلية local HTTP web service تعرض عملية GET operation عبر عنوان النسبي relative URI the **/api/todoitems/** يمكن للتطبيق الذي يعمل في iOS Simulator استهلاك العملية عن طريق إرسال طلب GET إلى `http://localhost:<port>/api/todoitems/`

يمكن للتطبيقات التي تعمل في محاكي Android الاتصال بخدمات ويب HTTP المحلية من خلال عنوان `10.0.2.2 address` هذا العنوان هو اسم مستعار لواجهة رد المضيف the host loopback interface (`127.0.0.1` على جهاز التطوير الخاص بك) يجب أيضاً إعداد تكوين أمان الشبكة network-security لعنوان IP المحدد هذا، على سبيل المثال، بالنظر إلى خدمة ويب HTTP محلية تعرض عملية GET عبر `/api/todoitems/ relative URI` يمكن لتطبيق يعمل في محاكي Android استهلاك العملية عن طريق إرسال طلب GET إلى `http://10.0.2.2:/api/todoitems/`

ملاحظة:

ASP. يجب على خدمات الويب ASP.NET Core التي يتم تشغيلها أثناء الاختبار على المضيف المحلي تعطيل عمليات إعادة التوجيه HTTPS من خلال التعليق على العبارة `app.UseHttpsRedirection();` في ملف Startup.cs

الكشف عن نظام التشغيل Detect the operating system

يمكن للتطبيق تحديد النظام الأساسي الذي يعمل عليه باستخدام DeviceInfo class في المثال التالي، يضبط التطبيق متغير BaseAddress variable على sets the قيمة مختلفة، اعتمادًا على ما إذا كان يعمل على نظام Android:

```
public static string BaseAddress = DeviceInfo.Platform ==
DevicePlatform.Android ? "http://10.0.2.2:5000" :
"http://localhost:5000";
public static string TodoItemsUrl =
$"{BaseAddress}/api/todoitems/";
```

٥ تمرين - استخدام خدمة REST service مع HttpClient

Azure Cloud Shell

إكمال هذا الدرس يتطلب توفر بيئة اختبار معزولة توفر لك بيئة الاختبار المعزولة إمكانية الوصول إلى موارد Azure لن يتم خصم أية تكاليف من اشتراكك لدى Azure يمكن استخدام بيئة الاختبار المعزولة فقط لإكمال التدريب على Microsoft Learn يُحظر الاستخدام لأي سبب آخر، وقد يؤدي إلى فقدان الوصول الدائم إلى صندوق الحماية.

توفر Microsoft التجربة العملية هذه والمحتوى ذي الصلة للأغراض التعليمية. تمتلك شركة Microsoft جميع المعلومات المقدمة وهي مخصصة فقط للتعرف على المنتجات والخدمات المغطاة في وحدة Microsoft Learn هذه.

تسجيل الدخول لتنشيط بيئة الاختبار المعزولة

كجزء من التطبيق الذي يستخدمه المهندسون في زيارات موقع العميل، تحتاج إلى إضافة ميزة تسمح للمهندس بالبحث عن تفاصيل المكونات الكهربائية. سيتم الاحتفاظ بهذه المعلومات في قاعدة بيانات والوصول إليها من خلال خدمة ويب REST كما طلب منك توفير واجهة تسمح للمسؤول بإنشاء تفاصيل الأجزاء الموجودة في قاعدة البيانات وإزالتها وتعديلها باستخدام نفس خدمة ويب REST

في هذا التمرين، ستقوم بنشر خدمة ويب REST إلى Azure ثم تحقق من أنه يمكنك الوصول إليها باستخدام مستعرض ويب، بعد ذلك، ستضيف وظائف إلى تطبيق موجود يستخدم خدمة ويب REST لاسترداد تفاصيل المكونات الكهربائية وإضافتها وحذفها وتحديثها.

ستقوم بتنفيذ هذا التمرين باستخدام بيئة الاختبار المعزولة ل Azure

لإكمال التدريب انتقال إلى تمرين - استخدام خدمة REST مع HttpClient

٦ الملخص

تستخدم العديد من التطبيقات الحديثة خدمات الويب REST web services لتوفير الوصول إلى البيانات أو ميزات أخرى مثل التخزين السحابي cloud storage

في .NET MAUI. يمكنك استخدام HttpClient لاستهلاك خدمات ويب REST وتنفيذ عمليات CRUD operations يحتوي .NET MAUI. أيضاً على واجهات برمجة التطبيقات التي يمكنك استخدامها للكشف عما إذا كان لديك اتصال إنترنت نشط، هذه المعلومات مهمة لأنه إذا قمت بتشغيل التعليمات البرمجية التي تستخدم الإنترنت دون وجود اتصال بالشبكة، فقد يتوقف تطبيقك عن الاستجابة.

وأخيراً، يتمتع كل من نظامي التشغيل iOS وAndroid بمجموعات شبكات أصلية يمكنك تكوينها لضمان اتصال تطبيقك بطريقة آمنة.

في هذه الوحدة، رأيت كيفية الاتصال بالإنترنت من داخل تطبيق .NET MAUI. واستخدام خدمة ويب REST وقد تعلمت عَلى وجه التحديد كيفية القيام بما يلي:

- اكتشف ما إذا كان جهازك متصلا بالإنترنت.
- استخدم خدمة ويب REST باستخدام HttpClient
- تكوين أمان شبكة العميل باستخدام ميزات الشبكات الأصلية لنظام التشغيل.

الوحدة الثامنة

تخزين البيانات المحلية باستخدام SQLite في تطبيق .NET MAUI

التعرف على كيفية تخزين البيانات الموجودة في SQLite والوصول إليها باستخدام تطبيق .NET MAUI.

الأهداف التعليمية

خلال هذه الوحدة، سوف تتمكن مما يلي:

- قارن بين خيارات تخزين البيانات المختلفة different data storage options المتوفرة لتطبيقات .NET MAUI.
- تخزين البيانات الارتباطية Store relational data في قاعدة بيانات SQLite
- التفاعل مع قاعدة البيانات بشكل غير متزامن database asynchronously للتأكد من أن واجهة المستخدم الخاصة بك تظل مستجيبة

محتويات الوحدة:

١- مقدمة

٢- مقارنة خيارات التخزين storage options

٣- تخزين البيانات محليًا باستخدام SQLite

٤- تمرين: تخزين البيانات محليًا باستخدام SQLite

٥- استخدام SQLite بشكل غير متزامن asynchronously

٦- تمرين: استخدام SQLite بشكل غير متزامن

٧- الملخص

١ المقدمة

عند إنشاء تطبيقات المحمول، من الشائع تخزين البيانات محليًا store data locally على جهاز معين لأسباب تتعلق بالأداء، يسمح لك .NET Multi-platform App UI (MAUI) بتخزين البيانات مؤقتاً محليًا على الجهاز الذي يتم تشغيل التطبيق عليه، يمكنك تخزين هذه البيانات باستخدام العديد من التقنيات المختلفة، لتخزين البيانات الارتباطية أو العلائقية store relational data يمكنك استخدام قاعدة بيانات SQLite database

لنفترض أنك تقوم بإنشاء تطبيق وسائط اجتماعية يتيح للمستخدمين الاتصال ببعضهم البعض، نظراً لأنه يتم تخزين البيانات المتعلقة بالمستخدمين عن بعد، يجب عليك استدعاء نقطة نهاية REST للحصول على معلومات حول المستخدمين الذين تتصل معهم في كل مرة تقوم فيها بإعادة تشغيل التطبيق.

ينجح عمل هذا التصميم، ومع ذلك، فهو غير فعال، لأنه يجب عليك الاستمرار في تنزيل معلومات المستخدم من الخادم في كل مرة تقوم فيها بإعادة تشغيل التطبيق.

يقوم التطبيق المصمم بكفاءة أكبر بتنزيل بيانات المستخدم مرة واحدة، وحفظها محليًا على الجهاز، باستخدام هذا التصميم، يجب عليك فقط تنزيل المعلومات للمستخدمين الآخرين، عندما تريد إجراء اتصال جديد، يمكنك إنشاء هذا التصميم الأكثر كفاءة باستخدام قاعدة بيانات محلية.

في هذه الوحدة، يمكنك إنشاء تطبيق .NET MAUI. الذي يخزن البيانات محليًا في قاعدة بيانات، تبدأ باستكشاف خيارات تخزين البيانات المختلفة المتوفرة.

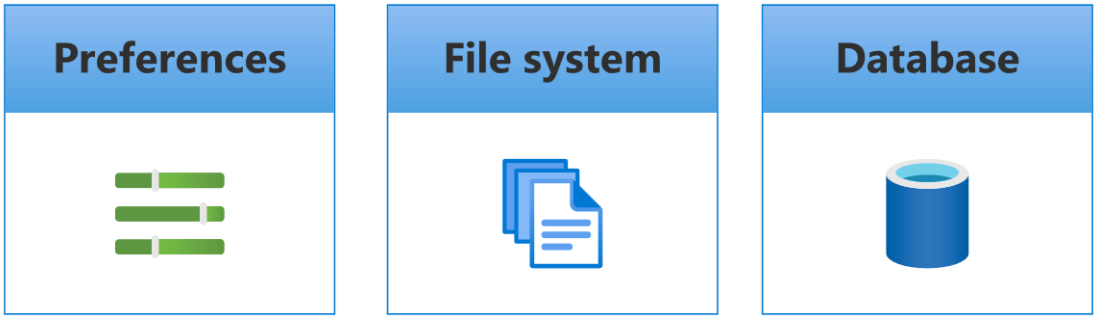
بعد ذلك، يمكنك إلقاء نظرة على SQLite وتعلم كيفية إدراج البيانات واستردادها. وأخيراً، يمكنك استخدام الأساليب غير المتزامنة the asynchronous methods المتوفرة في SQLite للتفاعل مع قاعدة البيانات دون حظر واجهة المستخدم.

بنهاية هذه الوحدة، يمكنك إنشاء تطبيق .NET MAUI. الذي يخزن البيانات محليًا في قاعدة بيانات SQLite

٢ مقارنة خيارات التخزين storage options

يوفر .NET Multi-platform App UI (MAUI) خيارات تخزين متعددة لتخزين البيانات مؤقتاً، محليًا على الجهاز، اعتمادًا على طبيعة البيانات وبنيتها وحجمها the nature, structure, and size of the data الخيارات الثلاثة الأكثر استخداماً لتخزين البيانات محليًا في تطبيق .NET MAUI هي:

- التفضيلات **Preferences** يخزن البيانات في أزواج قيم key-value pairs
- نظام الملفات **File system** يخزن الملفات غير المخزنة مباشرة على الجهاز من خلال الوصول إلى نظام الملفات file-system
- قاعدة البيانات **Database** تخزين البيانات في قاعدة بيانات ارتباطية a relational database



في هذا الدرس، ننظر إلى خيارات التخزين هذه والحالات التي يكون فيها كل خيار هو الأنسب.

متى تستخدم التفضيلات «Preferences»

تكون Preferences ملائمة عندما تعمل على أجزاء بسيطة من البيانات مثل تحديدات المستخدم، تُستخدم غالبًا للسماح للمستخدمين بتكوين التطبيق، يمكنك تخزين هذه البيانات كمجموعة من أزواج المفتاح/القيمة set of key/value pairs على سبيل المثال، افترض أنك تريد أن يتمكن المستخدم من تحديد ما إذا كان يجب على التطبيق حفظ اسم المستخدم وكلمة المرور بين جلسة العمل، يمكنك تخزين اختيار المستخدم في التفضيلات Preferences

توضح التعليمات البرمجية التالية كيفية تخزين قيمة منطقية Boolean value مسجلة في المتغير saveLoginDetails إلى preference يسمى SaveLogin ثم قراءة هذه البيانات مرة أخرى لاحقاً، لاحظ أن الأسلوب Get يتوقع منك توفير قيمة افتراضية إذا لم يتم العثور على التفضيل المحدد:

```
bool saveLoginDetails = ...;
...
Preferences.Set("SaveLogin", saveLoginDetails);
...
```

```
var savedPreference = Preferences.Get("SaveLogin", false);
```

تحتوي الفئة class `Preferences` أيضاً على أساليب لتحديد ما إذا كان التفضيل preference المسمى موجودًا (ContainsKey) وحذف تفضيل preference (Remove) وإزالة كافة بيانات التفضيل (Clear)

ملاحظة:

يجب تخزين أنواع البيانات البسيطة فقط كتفضيلات simple data types as preferences لا يمكنك تخزين مراجع references إلى كائنات objects كبيرة مثل القوائم والمجموعات والمصفوفات lists, collections, arrays بالنسبة لهذا النوع من البيانات، استخدم نظام الملفات file system أو قاعدة البيانات database

متى تستخدم نظام الملفات «File System»

تحتوي الأجهزة المحمولة وأجهزة سطح المكتب على نظام ملفات به بنية دليل تسلسل هرمي من المجلدات والملفات، من الملائم استخدام نظام الملفات عندما يكون لديك ملفات غير متماسكة مثل ملفات XML أو ملفات ثنائية أو ملفات نصية، على سبيل المثال، افترض أنك تريد تخزين بيانات السجل محليًا على الجهاز، يمكنك إنشاء ملف نصي، وحفظ هذا الملف إلى نظام الملفات وكتابة السجلات إليه عند حدوث الأحداث events يمكنك أيضاً تسلسل بنيات البيانات data structures الكبيرة إلى ملف، وتخزينها مؤقتًا محليًا على نظام الملفات إذا كنت بحاجة إلى الحفظ عند إيقاف تشغيل التطبيق، عند إعادة تشغيل التطبيق، يمكنك إعادة قراءة هذه البيانات مرة أخرى في الذاكرة.

تعرض التعليمات البرمجية التالية أمثلة لتسلسل البيانات إلى ملف وحفظ هذا الملف، ثم قراءة البيانات مرة أخرى، وإلغاء تسلسلها مرة أخرى في الذاكرة لاحقاً، هنا، نستخدم JSON لتنسيق التسلسل the serialization format ولكن يمكنك حفظ البيانات بأي تنسيق تشعر أنه الأنسب لطبيعة البيانات ومتطلبات أمان التطبيق.

```
using System.Text.Json;
using System.IO;

// Data to be written to the file system, and read back later
List<Customer> customers = ...;

// Serialize and save
string fileName = ...;
```

```
var serializedData = JsonSerializer.Serialize(customers);
File.WriteAllText(fileName, serializedData);
...

// Read and deserialize
var rawData = File.ReadAllText(fileName);
customers =
JsonSerializer.Deserialize<List<Customer>>(rawData);
```

الوصول إلى بيئة الاختبار المعزولة للتطبيق Access the app sandbox

عندما تعمل مع ملفات فضفاضة مثل ملفات XML تحتاج إلى تخزينها في موقع مناسب في نظام الملفات the file system قد تكون بعض هذه البيانات حساسة، ولا تريد حفظها في موقع يمكن للتطبيقات أو المستخدمين الآخرين الوصول إليه بسهولة، توفر تطبيقات .NET MAUI بيئة الاختبار المعزولة للتطبيق the app sandbox بيئة الاختبار المعزولة للتطبيق هي منطقة خاصة يمكن لتطبيقك العمل بها، بشكل افتراضي، لا يمكن لأي تطبيقات أخرى الوصول إلى هذه المنطقة بخلاف نظام التشغيل، يمكنك الوصول إلى بيئة الاختبار المعزولة باستخدام الخاصية الثابتة AppDataDirectory static للفئة FileSystem class

```
string path = FileSystem.AppDataDirectory;
```

في هذه التعليمات البرمجية يحتوي المتغير path على مسار الملف، إلى الموقع حيث يمكنك تخزين الملفات للتطبيق لاستخدامها، يمكنك قراءة البيانات وكتابتها إلى الملفات الموجودة في هذا المجلد، باستخدام التقنيات الموضحة في قسم متى تستخدم نظام الملفات.

ملاحظة:

الخاصية FileSystem.AppDataDirectory هي تجريد لمسار خاص بالجهاز؛ يتم تقييمها إلى مجلدات مختلفة على Android وiOS وWinUI3 يسمح لك هذا التجريد بكتابة التعليمات البرمجية التي تشير إلى بيئة الاختبار المعزولة بطريقة مستقلة عن نظام التشغيل، الذي يتم تشغيله عليه، استخدم هذا التجريد بدلاً من الرجوع إلى مسار خاص بالجهاز بشكل صريح في التعليمات البرمجية الخاصة بك.

إرشادات لحفظ البيانات في تطبيقات iOS

تمتلك Apple إرشادات iOS عن المكان الذي ينبغي تخزين الملفات فيه، هناك موقعان رئيسيان للمجلد:

- مجلد المكتبة The Library folder كما هو موضح سابقًا، تقوم الخاصية FileSystem.AppDataDirectory بإرجاع هذا المجلد، استخدم مجلد المكتبة the Library folder عند تخزين البيانات التي تم إنشاؤها بواسطة التطبيق.
- مجلد المستندات The Documents folder يوضح مقتطف التعليمات البرمجية التالي كيفية الإشارة إلى هذا المجلد في المتغير docFolder استخدم the Documents folder لتخزين البيانات التي أنشأها المستخدم فقط -user generated data يتم إنشاء هذه البيانات استجابة مباشرة لإجراء المستخدم، على سبيل المثال، إذا كنت تقوم بإنشاء تطبيق لتحرير النص مثل Microsoft Word فيمكنك تخزين المستند في مجلد المستندات the Documents folder

```
string docFolder =
Environment.GetFolderPath(Environment.SpecialFolder.
MyDocuments);
```

متى تستخدم قاعدة البيانات «Database»

من المستحسن استخدام قاعدة بيانات محلية عندما تكون لديك علاقات بين البيانات relationships between data أو عندما تريد تصفية البيانات filter the data بمرور الوقت، على سبيل المثال: في سيناريو الوسائط الاجتماعية، يحتوي كل منشور على بيانات حول المنشور، مثل الطابع الزمني والمحتويات، ومع ذلك، فإن كل منشور له علاقة أيضًا مع المستخدم الذي كتب المنشور، ومن المنطقي أن نمثل هذه العلاقة في قاعدة بيانات، لمنع ازدواجية البيانات بين المشاركات، وكذلك لتحسين كفاءة البحث عن البيانات.

قاعدة بيانات SQLite هي ملف، وتحتاج إلى تخزينها في مكان مناسب، من الناحية المثالية، يجب إنشاء مجلد ضمن المجلد `AppDataDirectory` في بيئة الاختبار المعزولة، وإنشاء قاعدة البيانات في هذا المجلد.

اختبار بسيط:

١- لنفترض أنك تقوم ببناء تطبيق .NET MAUI. لنظام iOS يتيح للمستخدمين إنشاء الموسيقى، تريد أن يتمكن المستخدمون من حفظ عملهم في ملف حتى يتمكنوا من إعادة تشغيل التطبيق ومتابعة العمل من حيث توقفوا، في بيئة الاختبار المعزولة لتطبيقات iOS في أي مجلد ستحفظ هذا الملف؟

٢- افترض أنك تريد تخزين جزء صغير من البيانات، بحيث يمكن الوصول إليها بين عمليات تشغيل التطبيقات، ما أنسب خيار لتخزين البيانات؟

حل الاختبار:

١- Documents

يمكنك استخدام مجلد المستندات the Documents folder لتخزين المحتوى الذي أنشأه المستخدم.

٢- Preferences

يتم استخدام التفضيلات Preferences لتخزين البيانات كأزواج قيم-مفتاح key-value هذا الموقع يشكّل موضعًا رائعًا للوحدات البسيطة من البيانات.

٣ تخزين البيانات محليًا باستخدام SQLite

يعد SQLite مفيدًا عندما يكون لديك بيانات ارتباطية أو علائقية relational data لنفترض أنك تقوم بإنشاء تطبيق وسائل التواصل الاجتماعي، تحتاج إلى تخزين معلومات حول المشتركين في التطبيق، تتضمن هذه البيانات معرفاً فريدًا لكل مستخدم واسمه، يمكنك بسهولة نمذجة هذا النوع من العلاقة في قاعدة بيانات SQLite

في هذا الدرس، ستتعلم كيفية استخدام SQLite في تطبيق .NET MAUI. باستخدام SQLite-net

ما SQLite

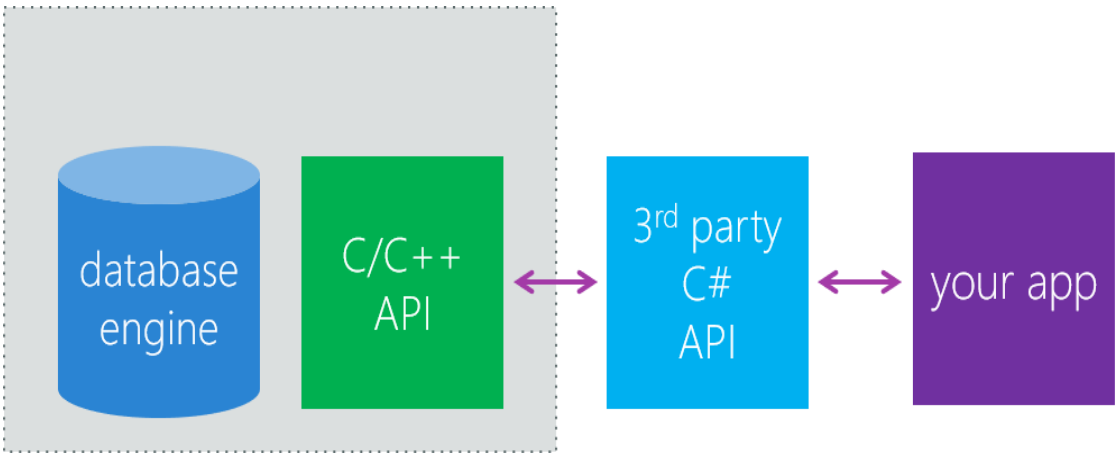
SQLite هي قاعدة بيانات محلية، خفيفة الوزن عبر الأنظمة، وهي معيار صناعي لتطبيقات الهاتف المحمول، لا يتطلب SQLite خادماً، يتم تخزين قاعدة البيانات في ملف قرص واحد على نظام ملفات الجهاز the device's file system يتم تشغيل جميع عمليات القراءة والكتابة مباشرةً على أساس ملف قرص SQLite

يتم تضمين مكتبات الأصلية SQLite native libraries في Android وiOS بشكل افتراضي؛ ومع ذلك، يدعم المحرك واجهة برمجة التطبيقات C/C++ API فقط، هذا السيناريو ليس مثالياً لمطوري .NET الذين يريدون طريقة معينة لتفاعل SQLite and .NET to interact

ما هو SQLite-net

هناك العديد من أدوات التغليف C# wrappers حول المحرك الأصلي the native SQLite engine الذي يمكن لمطوري .NET استخدامه، يستخدم العديد من مطوري .NET برنامج تغليف أو تضمين C# wrappers شائع يسمى SQLite-net

SQLite-net هو مخطط كائن علائقي object-relational يساعد على تبسيط عملية تعريف مخططات قاعدة البيانات defining database schemas من خلال السماح لك باستخدام النماذج the models المحددة في مشاريعك لتكون بمثابة المخطط the schema



على سبيل المثال، ضع في اعتبارك الفئة class التالية التي تقوم بإنشاء نموذج models a `User`

```
class User
{
    public int Id { get; set; }
    public string Username { get; set; }
    ...
}
```

باستخدام معين كائن علائقي object-relational يمكنك أخذ هذه الفئة الأولية initial `User` class وإنشاء جدول قاعدة بيانات يسمى User والذي يحتوي على أعمدة لحقلي `Id` and `Username` في هذه الفئة.

يتم توفير SQLite-net كحزمة NuGet package يجب عليك إضافة حزمة sqlite-net-pcl package إلى تطبيقاتك لاستخدامها.

كيفية الاتصال بقاعدة بيانات SQLite

يمكنك إنشاء اتصال بقاعدة بيانات SQLite من تطبيق من خلال كائن SQLiteConnection object يتم تعريف هذه الفئة في مساحة الاسم SQLite namespace جنبا إلى جنب مع الأنواع والأساليب الأخرى other types and methods التي يوفرها SQLite عند إنشاء مثيل لهذا الكائن، يمكنك تمرر اسم الملف لملف قاعدة البيانات، ثم يقوم المنشئ بفتح الملف إذا كان موجودًا، أو إنشائه إذا لم يكن موجودًا.

تعرض التعليمات البرمجية التالية مثالاً.

```
using SQLite;
...
string filename = ...
SQLiteConnection conn = new SQLiteConnection(filename);
```

تذكر، يجب أن يشير `filename` إلى موقع في بيئة الاختبار المعزولة للتطبيق.

كيفية إنشاء جدول create a table

تذكر أن SQLite-net هو مُعَيِّن كائنات العلاقات object-relational mapper ما يعني أنه يمكنك إنشاء مخطط قاعدة البيانات database schema من فئات C# classes يمكن لـ SQLite-net إنشاء جدول قاعدة بيانات من فئة C# class عادية، ولكن هناك العديد من السمات التي يمكنك إضافتها إلى فئة class لتوفير المزيد من بيانات التعريف أو الوصفية metadata تساعد metadata البيانات الوصفية SQLite على فرض ميزات مثل التفرد، وتطبيق القيود على بياناتك.

تتضمن السمات المتوفرة ما يلي:

- **Table** حدد اسم الجدول إذا كنت تريد أن يكون شيئاً آخر غير اسم الفئة the class's name
- **PrimaryKey** حدد أن العمود column هو المفتاح الأساسي the primary key
- **AutoIncrement** حدد أن العمود column يجب أن يزيد قيمته تلقائيًا، عند إدراج صف جديد new row is inserted
- **Column** حدد اسم العمود column إذا كنت تريد أن يكون شيئاً آخر غير اسم الخاصية the property name
- **MaxLength** حدد الحد الأقصى لعدد الأحرف characters التي يمكن استخدامها في العمود the column
- **Unique** حدد أن القيمة في العمود يجب أن تكون فريدة unique من جميع الصفوف rows الأخرى.

تعرض التعليمات البرمجية التالية إصداراً محدثاً من الفئة User class التي تطبق هذه السمات attributes

```
[Table("user")]
public class User
{
    // PrimaryKey is typically numeric
    [PrimaryKey, AutoIncrement, Column("_id")]
    public int Id { get; set; }

    [MaxLength(250), Unique]
    public string Username { get; set; }
    ...
}
```

بعد تعريف فئة C# class قم باستدعاء الأسلوب العام generic CreateTable method على الفئة SQLiteConnection class لإنشاء الجدول في قاعدة البيانات، حدد الفئة كمعلمة النوع the type parameter إليك مثال:

```
SQLiteConnection conn = new SQLiteConnection(filename);
conn.CreateTable<User>();
```

إذا كان الجدول the table موجودًا بالفعل في قاعدة البيانات، يتحقق الأسلوب CreateTable method من المخطط the schema لمعرفة ما إذا كانت هناك أي تغييرات، إذا كان هناك، تحاول العملية تحديث مخطط قاعدة البيانات update the database schema

كيفية تنفيذ عمليات القراءة والكتابة الأساسية basic read and write operations

بعد إنشاء جدول create a table يمكنك بدء التفاعل معه. لإضافة صف add a row استخدم الأسلوب Insert method على المثيل SQLiteConnection وقم بتوفير كائن object من النوع المناسب الذي يحتوي على البيانات المراد إدراجها، توضح التعليمات التالية كيفية إضافة صف جديد add a new row إلى الجدول User table

```
public int AddNewUser(User user)
{
    int result = conn.Insert(user);
    return result;
}
```

يقوم الأسلوب Insert method بإرجاع int والذي يمثل عدد الصفوف المدرجة في الجدول، في هذه الحالة، يكون العدد هو واحد.

لاسترداد صفوف من جدول، استخدم الأسلوب Table method يقوم هذا الأسلوب بإرجاع مجموعة من الكائنات objects (التي قد تكون فارغة empty):

```
public List<User> GetAllUsers()
{
   List<User> users = conn.Table<User>().ToList();
   return users;
}
```

يُعيد الأسلوب Table method كائن `TableQuery\<T> object` للحصول على List استخدم الأسلوب ToList method كما هو موضح في المثال السابق.

تنفيذ استعلام Execute a SQLite query باستخدام LINQ

يسترد الأسلوب Table method كافة الصفوف من جدول rows from a table في معظم الحالات، تريد إرجاع فقط مجموعة فرعية من الصفوف subset of the rows التي تطابق مجموعة من المعايير المحددة. لهذه المهام، استخدم LINQ مع SQLite-net.

يدعم SQLite-net العديد من استعلامات LINQ queries الشائعة بما في ذلك:

- `Where`
- `Take`
- `Skip`
- `OrderBy`
- `OrderByDescending`
- `ThenBy`
- `ElementAt`
- `First`
- `FirstOrDefault`
- `ThenByDescending`
- `Count`

باستخدام هذه الوسائل، يمكنك استخدام بنية أسلوب التوسيع the extension method syntax أو بنية the LINQ C# syntax على سبيل المثال، إليك مقطع من التعليمات التي تمكنك من استرداد تفاصيل مستخدم محدد:

```
public User GetByUsername(string username)
{
    var user = from u in conn.Table<User>()
               where u.Username == username
               select u;
    return user.FirstOrDefault();
}
```

تحديث الصفوف وحذفها Update and delete rows

يمكنك تحديث صف باستخدام update a row يمكنك the `SQLiteConnection` object's `Update` method تقديم كائن يحدد الصف ليتم تحديثه بقيمه جديدة new values يعدل الأسلوب `Update` method الصف الذي يحتوي على نفس قيمة المفتاح الأساسي primary key value للكائن المقدم the provided object القيمة التي تم إرجاعها هي عدد الصفوف rows التي تم تغييرها. إذا كانت هذه القيمة صفراً، فهذا يعني أنه لم يتم العثور على أي صفوف بمفتاح أساسي مطابق، ولم يتم تحديث أي شيء، يوضح المقطع التالي هذه الطريقة أثناء العمل:

```
public int UpdateUser(User user)
{
    int result = 0;
    result = conn.Update(user);
    return result;
}
```

إزالة الصفوف من جدول باستخدام the `SQLiteConnection` object's `Delete` method الشكل الأبسط لهذا الأسلوب يأخذ المفتاح الأساسي للعنصر المراد حذفه كمعلمة، كما هو موضح في المثال التالي، هذا هو الشكل من أسلوب الحذف عام the `Delete` method is generic ويتطلب معلمة نوع type parameter القيمة التي تم إرجاعها The value returned هي عدد الصفوف التي تمت إزالتها the number of rows removed من الجدول the table

```
public int DeleteUser(int userID)
{
    int result = 0;
    result = conn.Delete<User>(userID);
    return result;
}
```

٤ تمرين: تخزين البيانات محليًا باستخدام SQLite

في هذا التمرين، يمكنك استخدام SQLite لتخزين المعلومات محليًا store information locally مع أحد التطبيقات، في السيناريو النموذجي، قررت تخزين البيانات مؤقتًا لتطبيق الوسائط الاجتماعية لتحسين الاستجابة، ينشئ هذا التمرين قاعدة بيانات SQLite محلية، ويستخدمها لتخزين المعلومات حول الأشخاص. يمكنك حفظ ملف قاعدة البيانات الفعلي في وحدة تخزين محلية.

تستخدم هذه الوحدة النمطية .NET 9.0 SDK. تأكد من تثبيت .NET 9.0.

افتح the starter solution

١- استنساخ أو تحميل exercise repo

٢- استخدم Visual Studio لفتح حل People.sln solution الذي تجده في **mslearn-dotnetmaui-store-local-data** > **People** أو مجلد the starter folder في Visual Studio Code

ملحوظة:

لا تحاول تشغيل التطبيق الآن، التعليمات البرمجية غير مكتملة، وستطرح استثناءات، حتى تقوم بإضافة العناصر المفقودة لاحقاً في هذا التمرين.

تعريف كيان Define a SQLite entity

١- افتح ملف Person.cs file في مجلد Models

٢- أضف خاصية int تسمى Id إلى Person class

٣- إضافة خاصية string تسمى Name يجب أن تبدو الفئة class كما يلي:

```
namespace People.Models;

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

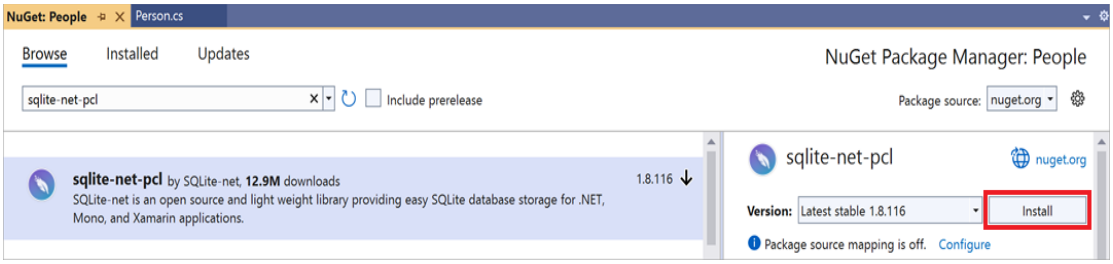
٤- احفظ ملف Person.cs

إضافة مكتبة Add the SQLite library

١- انقر بزر الماوس الأيمن فوق عقدة مشروع People project من مستكشف الحلول Solution Explorer في Visual Studio

٢- في قائمة السياق the context menu التي تظهر، حدد Manage NuGet Packages

٣- ابحث عن sqlite-net-pcl وحدده، ثم حدد Install



إذا كنت تستخدم Visual Studio Code فافتح المحطة الطرفية the terminal وهذه الحزم packages باستخدام الأوامر التالية:

```
dotnet add package sqlite-net-pcl
```

إضافة سمات Add SQLite attributes

١- في ملف Person.cs أضف توجيها `using directive` لمساحة الاسم `SQLite` namespace إلى ملف Person class يتيح لك هذا التوجيه استخدام سمات the SQLite attributes

```
using SQLite;

namespace People.Models;

public class Person
{
    ...
}
```

٢- قم بتفسير أو توضيح Annotate the Person class بالسمة `[Table]` attribute وحدد اسم الجدول ك people

٣- حدد الخاصية Id كمفتاح أساسي the primary key قم بتفسيرها مع السمتين [PrimaryKey] and [AutoIncrement] attributes

٤- إضافة تفسيرات إلى الخاصية Add annotations to the Name property حدد الحد الأقصى لطولها على 250 as `MaxLength` حدد أن تكون كل قيمة في العمود value in the column فريدة `Unique`

يجب أن تبدو الفئة class المكتملة كما يلي:

```
using SQLite;

namespace People.Models;

[Table("people")]
public class Person
{
    [PrimaryKey, AutoIncrement]
    public int Id { get; set; }

    [MaxLength(250), Unique]
    public string Name { get; set; }
}
```

٥- احفظ ملف Person.cs

الاتصال بقاعدة البيانات Connect to the database

١- افتح ملف PersonRepository.cs

٢- افحص الفئة PersonRepository class تحتوي هذه الفئة على تعليمات برمجية هيكلية غير مكتملة مع علامات TODO حيث تضيف الوظيفة للوصول إلى قاعدة البيانات.

٣- أضف توجيها using directive لمساحات الأسماء SQLite and People.Models namespaces إلى ملف الفئة PersonRepository.cs class

٤- أضف حقلاً خاصاً private SQLiteConnection باسم conn إلى الفئة، أعلى الدالة the Init function

٥- في الدالة the `Init` function تأكد أن conn لا يساوي null إذا كان الأمر كذلك، فارجع على الفور.

```
if (conn != null)
    return;
```

بهذه الطريقة، يتم تشغيل التعليمة البرمجية للتهيئة لقاعدة بيانات SQLite database مرة واحدة فقط.

٦- تهيئة الحقل conn للاتصال بقاعدة البيانات باستخدام المتغير _dbPath variable

٧- استخدم الأسلوب conn.CreateTable method لإنشاء جدول create a table لتخزين بيانات Person يجب أن تبدو الدالة Init المكتملة كما يلي:

```
using SQLite;
using People.Models;
...

private SQLiteConnection conn;
...
private void Init()
{
   if (conn != null)
      return;

   conn = new SQLiteConnection(_dbPath);
   conn.CreateTable<Person>();
}
```

إدراج صف في قاعدة البيانات Insert a row into the database

١- في الفئة PersonRepository class ابحث عن الأسلوب AddNewPerson method

٢- لإدراج كائن جديد Person object استبدل التعليق TODO في هذا الأسلوب بالتعليمات البرمجية، تستدعي التعليمات البرمجية `Init` أولاً للتحقق من تهيئة قاعدة البيانات the database is initialized ثم تستخدم the `SQLiteConnection` object's `Insert` method تعيين المتغير `result` إلى القيمة التي يرجعها الأسلوب `Insert` كما هو موضح في التعليمات البرمجية التالية:

```
public void AddNewPerson(string name)
{
   int result = 0;
   try
   {
      // enter this line
      Init();

   // basic validation to ensure a name was entered
     if (string.IsNullOrEmpty(name))
       throw new Exception("Valid name required");

     // enter this line
 result = conn.Insert(new Person { Name = name });
      ...
   }
   ...
}
```

استرداد الصفوف من قاعدة البيانات Retrieve rows from the database

١- في الفئة PersonRepository class ابحث عن الأسلوب GetAllPeople

٢- استدعي Init للتأكد من تهيئة قاعدة البيانات.

٣- استخدم الأسلوب العام the generic Table\<T> method لاسترداد كافة الصفوف في الجدول، حدد Person كمعلمة النوع the type parameter

٤- استخدم أسلوب الملحق the ToList() extension method لتحويل النتيجة إلى مجموعة List\<Person> collection وإرجاع هذه المجموعة collection

٥- أضف معالجة الأخطاء بتغليف التعليمات البرمجية في كتلة try-catch block إذا كان هناك خطأ، فقم بتعيين الخاصية StatusMessage إلى خاصية الاستثناء the exception's Message وإرجاع مجموعة فارغة empty collection يجب أن يبدو الأسلوب المكتمل كما يلي:

```
public List<Person> GetAllPeople()
{
   try
   {
      Init();
      return conn.Table<Person>().ToList();
   }
   catch (Exception ex)
   {
      StatusMessage = string.Format("Failed to
retrieve data. {0}", ex.Message);
   }

   return new List<Person>();
}
```

٦- احفظ ملف PersonRepository.cs

Integrate the repository into the UI دمج المستودع في واجهة المستخدم

١- افتح ملف MauiProgram.cs

٢- في الدالة CreateMauiApp function بعد العبارات التي تضيف الصفحة MainPage page كخدمة أحادية إلى التطبيق، أضف التعليمات البرمجية لتنفيذ المهام التالية:

- أنشئ متغير سلسلة string variable باسم dbPath قم بتهيئة هذه السلسلة باستخدام التعبير FileAccessHelper.GetLocalFilePath("people.db3") يسمى ملف قاعدة البيانات الذي يستخدمه التطبيق people.db3 ويحفظ التطبيق هذا الملف في وحدة تخزين محلية على الجهاز.
- استخدم إدخال التبعية لإضافة الفئة PersonRepository class كخدمة أحادية إلى التطبيق، تعرض الفئة PersonRepository class منشئًا constructor يأخذ المسار إلى ملف قاعدة البيانات كمعلمة سلسلة string parameter

يجب أن تبدو التعليمات البرمجية المكتملة للدالة the CreateMauiApp function كما يلي:

```
public static MauiApp CreateMauiApp()
{
  var builder = MauiApp.CreateBuilder();
  builder
      .UseMauiApp<App>()
      .ConfigureFonts(fonts =>
      {
          fonts.AddFont("OpenSans-Regular.ttf",
"OpenSansRegular");
          fonts.AddFont("OpenSans-Semibold.ttf",
"OpenSansSemibold");
      });

  // Add this code
  string dbPath =
FileAccessHelper.GetLocalFilePath("people.db3");

builder.Services.AddSingleton<PersonRepository>
(s =>
```

```
ActivatorUtilities.CreateInstance<PersonReposit
ory>(s, dbPath));

  return builder.Build();
}
```

٣- احفظ ملف MauiProgram.cs

٤- قم بتوسيع App.xaml في Solution Explorer ثم افتح ملف App.xaml.cs

٥- أضف خاصية public, static تسمى `PersonRepo` تحتفظ هذه الخاصية بكائن PersonRepository object إلى الفئة App class

٦- تهيئة الخاصية PersonRepo في المنشئ/الدالة الإنشائية constructor عن طريق إضافة معلمة PersonRepository parameter إلى الدالة الإنشائية، وتعيين خاصية 'PersonRepo' إلى القيمة الموجودة في هذه المعلمة.

يجب أن تبدو فئة App class المكتملة كما يلي:

```
public partial class App : Application
{
  public static PersonRepository PersonRepo { get;
private set; }

  public App(PersonRepository repo)
  {
      InitializeComponent();
      PersonRepo = repo;
  }
}
```

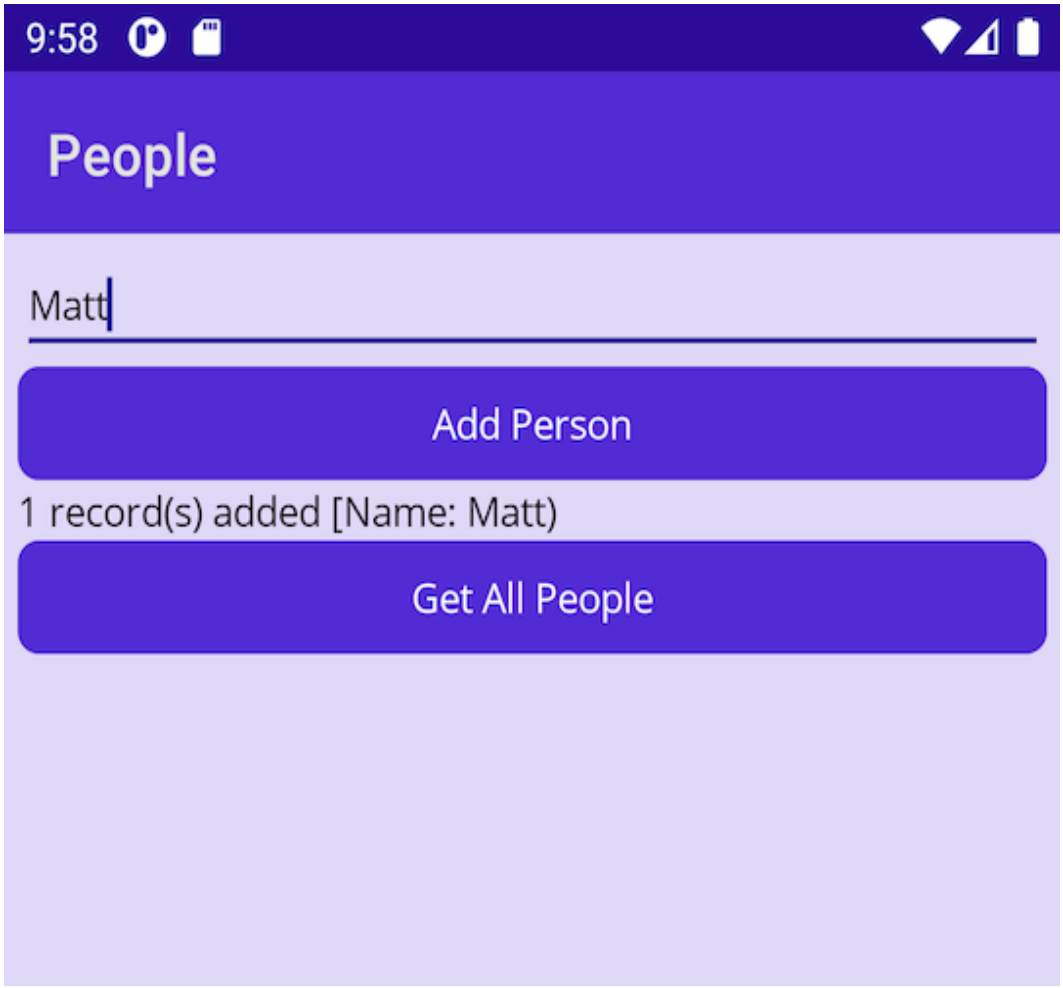
ملحوظة:

تقوم عملية إدخال التبعية The dependency injection process تلقائياً بملء المعلمة repo parameter إلى الدالة الإنشائية.

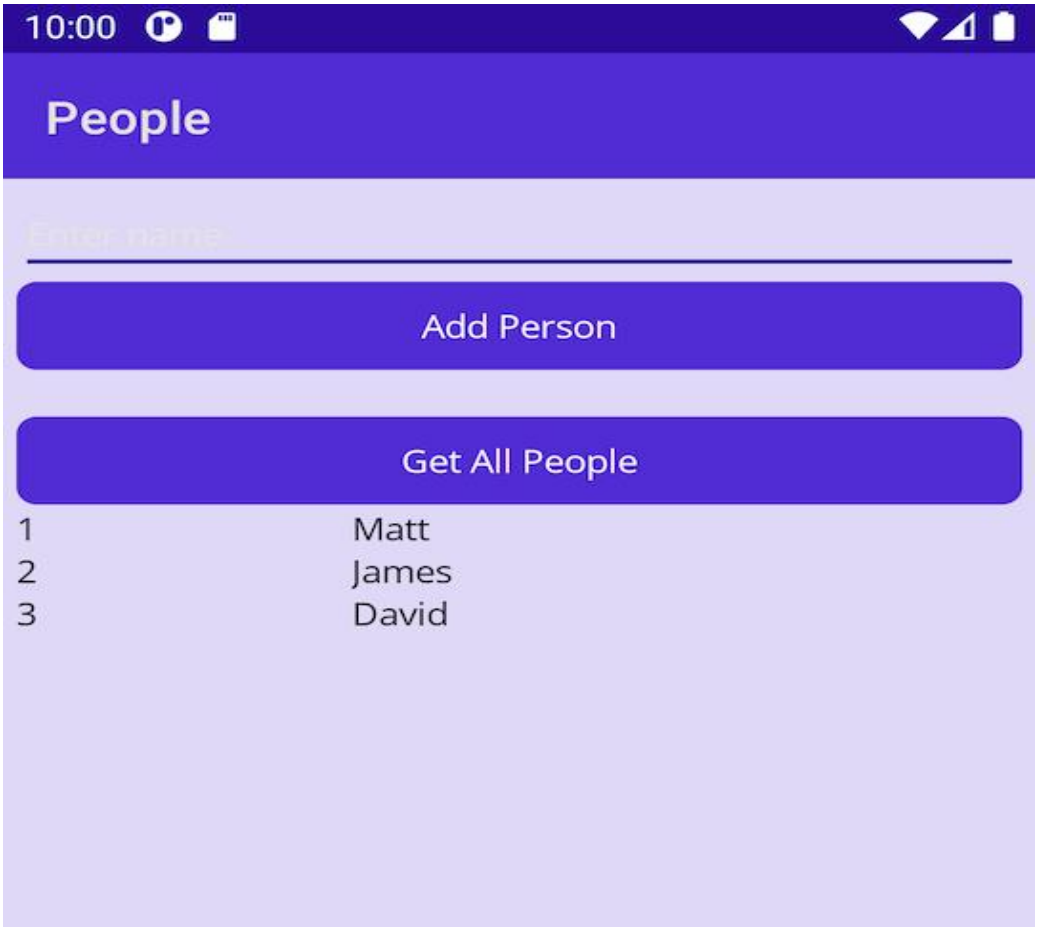
اختبر التطبيق

١- بناء الحل Build the solution باستخدام CTRL+Shift+B

٢- بمجرد اكتمال البناء، ابدأ تصحيح الأخطاء start debugging باستخدام F5 عند ظهور واجهة المستخدم، أدخل اسمك وحدد Add Person



٣- حدد Get All People وتحقق من ظهور اسمك.



٤- جرب عن طريق إضافة المزيد من الأسماء واسترداد قائمة الأشخاص المخزنين.

٥- ارجع إلى Visual Studio أو Visual Studio Code وتوقف عن تصحيح الأخطاء باستخدام Shift+F5

٦- أعد تشغيل التطبيق، وحدد Get All People تحقق من أن الأسماء التي قمت بتخزينها مسبقاً، لا تزال مخزنة في قاعدة البيانات. أغلق التطبيق عند الانتهاء.

٥ استخدام SQLite بشكل غير متزامن asynchronously

إذا قمت بتشغيل الاستعلامات على قاعدة بيانات بطريقة متزامنة synchronous فقد يؤدي ذلك إلى مشاكل في الأداء وتطبيقات غير مستجيبة، يحتوي SQLite-net على واجهة برمجة تطبيقات غير متزامنة an asynchronous API يمكنك استخدامها للحفاظ على استجابة تطبيقك.

في هذا الدرس، ستتعلم كيفية استخدام واجهة برمجة تطبيقات غير المتزامنة SQLite-net asynchronous API لضمان بقاء تطبيقك مستجيبًا بدرجة عالية.

فهم الاستعلامات غير المتزامنة Understand asynchronous queries

حتى هذه النقطة، قمت بتنفيذ جميع عمليات قاعدة البيانات على مؤشر ترابط واجهة المستخدم database operations on the UI thread ومع ذلك، لبناء تطبيق هاتف عالي الاستجابة، قد ترغب في فعل الأمور بشكل مختلف قليلاً، إذا شغّلت عمليات قاعدة بيانات على مؤشر ترابط واجهة المستخدم، فقد يتسبب في تجميد واجهة المستخدم إذا استغرقت العملية وقتًا طويلاً لاستكمالها.

لحل هذه المشكلة، يتضمن SQLite-net واجهة برمجة تطبيقات غير متزامنة asynchronous API من خلال الفئة class `SQLiteAsyncConnection` على سبيل المثال، لإنشاء جدول بشكل غير متزامن create a table asynchronously يمكنك استخدام الأسلوب method `CreateTableAsync`

```
var conn = new SQLiteAsyncConnection(dbPath);
await conn.CreateTableAsync<User>();
```

تنفيذ عمليات غير متزامنة async operations باستخدام SQLite-net

تعرض الفئة class `SQLiteAsyncConnection` عمليات مماثلة لنظيرتها المتزامنة. ومع ذلك، فإن العمليات تعتمد جميعها على المهام للاستخدام في الخلفية.

تتضمن العمليات غير المتزامنة asynchronous operations الشائعة المتوفرة ما يلي:

- **CreateTableAsync** إنشاء جدول Creates a table بناءً إلى الفئة المشار إليها the indicated class
- **DropTableAsync** حذف الجدول Drops the table الذي يرتبط بالفئة المشار إليها the indicated class
- **GetAsync** يحصل على السجل Gets the record في الجدول الذي يرتبط بالفئة المشار إليها، ويطابق المفتاح الأساسي the primary key الذي تم تمريره إلى الدالة الإنشائية the constructor

- **InsertAsync** إدراج سجل جديد Inserts a new record باستخدام العنصر the item الذي تم تمريره إلى الدالة الإنشائية the constructor
- **UpdateAsync** تحديث سجل موجود Updates an existing record باستخدام العنصر the item الذي تم تمريره إلى الدالة الإنشائية the constructor
- **DeleteAsync** يحذف السجل Deletes the record في الجدول الذي يتوافق مع الفئة المشار إليها، ويطابق المفتاح الأساسي the primary key الذي تم تمريره إلى الدالة الإنشائية
- **QueryAsync** تشغيل استعلام SQL query مباشر، وإرجاع كائن returns an object
- **ExecuteAsync** تشغيل استعلام SQL query مباشر، وإرجاع عدد الصفوف المتأثرة affected rows
- **ExecuteScalarAsync** تشغيل استعلام SQL query مباشر، وإرجاع النتيجة الفردية the single result
- **ToListAsync** تنفيذ أسلوب الجدول Executes the Table method بشكل غير متزامن asynchronously

توضح التعليمات البرمجية التالية مثالاً على كيفية استخدام الأسلوب `ToListAsync` method لاسترداد السجلات بشكل غير متزامن:

```
SQLiteAsyncConnection conn;
ObservableCollection<User> userList;// Bound to UI
...
public async Task AddAllUsersAsync()
{
List<User> users = await conn.Table<User>().ToListAsync();
    // Must be on UI thread here!
    foreach (var u in users)
        userList.Add(u);
}
```

في هذا المثال، يجلب الأسلوب `ToListAsync` جميع المستخدمين users من قاعدة البيانات بشكل غير متزامن the database asynchronously إذا كنت تستخدم هذا الأسلوب، تظل واجهة المستخدم الخاصة بك مستجيبة حتى إذا كانت هناك مجموعة كبيرة من المستخدمين في قاعدة البيانات.

٦ تمرين: استخدام SQLite بشكل غير متزامن

يعمل التطبيق بشكل جيد، ولكن إذا كانت قاعدة البيانات تحتوي على العديد من الصفوف، يمكن أن تصبح واجهة المستخدم غير مستجيبة بينما يقوم التطبيق بإجراء استعلامات قاعدة البيانات، والعمليات الأخرى. في هذا التمرين، يمكنك تحويل التطبيق من واجهة برمجة التطبيقات المتزامنة the synchronous SQLite API إلى الإصدار غير المتزامن the asynchronous بهذه الطريقة، يستجيب تطبيقك دائماً بغض النظر عن عدد الاستعلامات التي تجريها على قاعدة البيانات الخاصة بك.

إنشاء اتصال غير متزامن Create an Async connection

١- افتح ملف PersonRepository.cs في the People project

٢- تعديل تعريف الأسلوب Init ليكون async قم بتغيير نوع إرجاع الأسلوب إلى Task

٣- قم بتغيير الخاصية conn إلى SQLiteAsyncConnection وقم بتحديث التعليمات البرمجية في الأسلوب Init الذي يقوم بتهيئة الاتصال connection

٤- استبدل الاستدعاء بالأسلوب المتزامن the synchronous CreateTable method بالأسلوب غير المتزامن the asynchronous CreateTableAsync method

تبدو التعليمة البرمجية المكتملة كما يلي:

```
private SQLiteAsyncConnection conn;

private async Task Init()
{
    if (conn != null)
        return;

    conn = new SQLiteAsyncConnection(_dbPath);

    await conn.CreateTableAsync<Person>();
}
```

إدراج عنصر في جدول بشكل غير متزامن Insert an item into a table asynchronously

١- تعديل تعريف الأسلوب `AddNewPerson` ليكون async قم بتغيير نوع إرجاع الأسلوب إلى `Task`

٢- أضف الكلمة الأساسية `await` keyword إلى استدعاء الأسلوب `Init` لأن Init الآن أصبح أسلوب `async` method

٣- قم بتحديث الأسلوب `AddNewPerson` لإدراج `Person` جديد، باستخدام عملية إدراج غير متزامنة asynchronous insert operation

يجب أن تبدو التعليمات البرمجية كهذه:

```
using System.Threading.Tasks;
...
public async Task AddNewPerson(string name)
{
   int result = 0;
   try
   {
      // Call Init()
      await Init();

   // basic validation to ensure a name was entered
    if (string.IsNullOrEmpty(name))
       throw new Exception("Valid name required");

   result = await conn.InsertAsync(new Person {
Name = name });

      StatusMessage = string.Format("{0} record(s)
added [Name: {1})", result, name);
   }
   catch (Exception ex)
   {
     StatusMessage = string.Format("Failed to add
{0}. Error: {1}", name, ex.Message);
   }
}
```

الحصول على جميع العناصر items من جدول بشكل غير متزامن table asynchronously

١- تعديل تعريف الأسلوب `GetAllPeople` يجب أن يكون `async` وإرجاع كائن `Task<List<Person>>` object

٢- أضف الكلمة الأساسية keyword `await` إلى استدعاء الأسلوب Init

٣- قم بتحديث الأسلوب لإرجاع النتائج باستخدام استدعاء غير متزامن asynchronous call

يجب أن تبدو التعليمات البرمجية كهذه:

```
public async Task<List<Person>> GetAllPeople()
{
   try
   {
     await Init();
  return await conn.Table<Person>().ToListAsync();
   }
   catch (Exception ex)
   {
      StatusMessage = string.Format("Failed to
retrieve data. {0}", ex.Message);
   }

   return new List<Person>();
}
```

٤- احفظ ملف PersonRepository.cs

اختبار الوظيفة غير المتزامنة Test the asynchronous functionality

١- قم بتوسيع MainPage.xaml في مستكشف الحلول Solution Explorer وافتح ملف MainPage.xaml.cs

٢- قم بتعديل كل من معالجات أحداث النقر بالزر button-click event handlers بحيث تستخدم الأساليب غير المتزامنة the asynchronous methods من الفئة `PersonRepository` class الاستفادة من الكلمات الأساسية `async` and `await` keywords

```
public async void OnNewButtonClicked(object sender,
EventArgs args)
  {
     statusMessage.Text = "";

   await App.PersonRepo.AddNewPerson(newPerson.Text);
     statusMessage.Text =
App.PersonRepo.StatusMessage;
  }

public async void OnGetButtonClicked(object sender,
EventArgs args)
  {
     statusMessage.Text = "";

    List<Person> people = await
App.PersonRepo.GetAllPeople();
     peopleList.ItemsSource = people;
  }
```

٤- احفظ ملف MainPage.xaml.cs

٥- قم ببناء وتشغيل البرنامج Build and run على Windows and Android وتأكد أنه يعمل كما كان من قبل.

٧ الملخص

يمكن أن يكون تخزين البيانات محليًا على جهاز محمول مفيدًا لتحسين الأداء، بدلاً من إجراء استدعاءات باستمرار إلى خادم بعيد للحصول على البيانات، يمكنك تخزين البيانات الهامة محليًا، واستردادها بسرعة.

اعتمادًا على نوع البيانات الموجودة لديك، هناك خيارات تخزين مختلفة متاحة، عندما تعمل مع بيانات ذات طبيعة علائقية، فإن قاعدة البيانات هي الخيار الأفضل.

يمكنك إنشاء قاعدة بيانات محلية في تطبيق .NET Multi-platform App UI (MAUI) app باستخدام SQLite-net ال SQLite-net هو برنامج تغلف أو تضمين C# wrapper around SQLite يعرض SQLite واجهة برمجة تطبيقات غير متزامنة asynchronous API للمساعدة في ضمان بقاء واجهة مستخدم التطبيق مستجيبة دائمًا.

في هذه الوحدة، تعلمت كيفية تخزين البيانات محلياً store data locally في تطبيق .NET MAUI. وقد تعلمت عَلى وجه التحديد كيفية القيام بما يلي:

- مقارنة بين خيارات تخزين البيانات المختلفة، المتوفرة لتطبيقات .NET MAUI.
- تخزين البيانات العلائقية Store relational data في قاعدة بيانات SQLite
- التفاعل مع قاعدة بيانات بشكل غير متزامن database asynchronously للتأكد من أن واجهة المستخدم الخاصة بك تظل مستجيبة.

معرفة المزيد **.NET MAUI documentation**

الوحدة التاسعة

تصميم نموذج عرض MVVM viewmodel لـ .NET MAUI.

تعرف على نمط تصميم MVVM وكيف يمكنه فصل منطق العمل، عن تعليمات واجهة المستخدم. تعرف على كيفية تصميم نماذج العرض viewmodels ولماذا تعد جزءًا مهمًا من النمط.

الأهداف التعليمية

في نهاية هذه الوحدة، ستتمكن من:

- تحديد متى تستخدم نمط the Model-View-ViewModel (MVVM) pattern
- استخدام نموذج عرض viewmodel لتوجيه السلوك المرئي visual behavior
- تحسين قابلية اختبار التعليمات البرمجية من خلال خصائص قابلة للربط بالبيانات data-bindable properties

محتويات الوحدة:

١- مقدمة

٢- ما هو نمط MVVM

٣- استخدام نموذج عرض viewmodel

٤- الملخص

١ المقدمة

The Model-View-ViewModel (MVVM) pattern هو نمط تصميم برمجي، أنماط التصميم عبارة عن مجموعات من القواعد والقوالب التي تساعد في جعل التعليمات البرمجية للتطبيق أفضل أو أكثر اتساقًا. يمكنك استخدام نمط the MVVM pattern للمساعدة في فصل منطق سلوك التطبيق عن تعليمات عرض واجهة المستخدم، ولتنفيذ ذلك بطريقة تدعم اختبار الوحدة supports unit testing

مثال افتراضي

لنفترض أن شركتك تنتج برنامج إدارة لأقسام الموارد البشرية، يستخدم البرنامج لإدارة ومراجعة التفاصيل المتعلقة بموظفي الشركة، النظام الحالي قائم على الويب، طُلب منك إنشاء تطبيق للهواتف يُمكن استخدامه مع نفس واجهات برمجة التطبيقات الخلفية التي يستخدمها نظام الويب، أنت تعلم أن واجهات برمجة التطبيقات الخلفية -the back end APIs وواجهة الويب الأمامية the web front end مغطاة بالكامل باختبارات الوحدة unit tests وتريد الإبقاء على هذا المعيار لتطبيق الهاتف. كما سمعت أن MVVM pattern يمكن أن يساعد في تصميم تطبيقات الهاتف المحمول، لتحقيق أقصى قدر من قابلية الاختبار، لذا تخطط لتجربته.

MVVM بحد ذاته هو نمط تصميم design pattern يشكل الأساس للعديد من الأطر وأدوات البرمجة frameworks and programming tool-kits كما توفر معظم هذه الأطر أشياء أخرى، مثل تجريدات التنقل والمراسلة navigation and messaging abstractions التي تساعد MVVM promote على تعزيز اختبار الوحدات unit testing ستركز على أساسيات the pattern في هذه المناقشة، لأن الأطر يمكن أن تختلف على نطاق واسع في تنفيذها.

ما ستفعله

في هذه الوحدة، ستتعرف على أجزاء the MVVM pattern وستتعلم the responsibilities of the model, view, and viewmodel ثم ستكتب التعليمات البرمجية في viewmodel للتعامل مع سلوك التطبيق بطريقة معزولة عن مكتبات واجهة المستخدم (حتى من .NET MAUI نفسه)

ما الذي تتعلمه

- تحدد متى تستخدم نمط the Model-View-ViewModel pattern
- تستخدم نموذج عرض لتوجيه السلوك المرئي viewmodel to drive visual behavior
- تحسين قابلية اختبار الكود من خلال خصائص قابلة للربط بالبيانات code testability through data-bindable properties

٢ ما هو MVVM

عادةً ما تحتوي تطبيقات .NET MAUI. التي لا تستخدم MVVM على المزيد من التعليمات في ملفات التعليمات البرمجية الخلفية The code-behind files الخاصة بها، تتبع ملفات التعليمات البرمجية الخلفية في .NET MAUI. النمط التالي: `{something}.xaml.cs` عادةً ما تتحكم معظم التعليمات في ملف التعليمات البرمجية الخلفية في سلوك واجهة المستخدم (UI) يتضمن سلوك واجهة المستخدم أي أمر يحدث لـ واجهة المستخدم، مثل تغيير لون أو نص ما، وقد يتضمن أي شيء يحدث بسبب واجهة المستخدم، بما في ذلك معالجات النقر على الأزرار button click handlers

تتمثل إحدى مشكلات هذا النهج في صعوبة كتابة اختبارات الوحدة unit tests على ملفات التعليمات البرمجية الخلفية code-behind files غالبًا ما تفترض ملفات التعليمات البرمجية الخلفية حالة تطبيق تم إنشاؤها بواسطة تحليل XAML أو حتى تم إنشاؤها بواسطة صفحات أخرى. يصعب التعامل مع هذه الظروف في برنامج تشغيل اختبار الوحدة، الذي قد لا يعمل حتى على جهاز محمول، ناهيك عن واجهة المستخدم، لذا، نادرًا ما تكون اختبارات الوحدة قادرة على اختبار سلوكيات واجهة المستخدم في هذه الملفات.

هنا يكون the MVVM pattern مفيدًا، فعند استخدامه بشكل صحيح، يحل the MVVM pattern هذه المشكلات عن طريق نقل معظم منطق سلوك واجهة المستخدم إلى فئات الوحدة القابلة للاختبار unit-testable classes التي تسمى viewmodels ويُستخدم The MVVM pattern بشكل شائع مع الأطر frameworks التي تدعم ربط البيانات data-binding وذلك لأنه باستخدام .NET MAUI يمكنك ربط البيانات بكل عنصر من عناصر واجهة المستخدم ب viewmodel وتجاهل التعليمات البرمجية أو تجاهلها تقريباً في طريقة عرض view أو التعليمات البرمجية في الخلف.

ما هي أجزاء تطبيق the parts of an MVVM

في حين أن viewmodel هو الجزء الفريد من the MVVM pattern يحدد النمط أيضاً model part and view part تتوافق تعريفات هذه الأجزاء مع بعض الأنماط الشائعة الأخرى، مثل Model-View-Controller (MVC)

ما هو model

في تطبيق MVVM application يتم استخدام مصطلح **model** للإشارة إلى بيانات العمل وعملياتك business data and operations لا يتدخل The model في عرض التطبيق للمستخدم.

القاعدة المفيدة لتحديد التعليمات البرمجية التي تنتمي إلى the model هي أنه يجب أن يكون قابلاً للنقل عبر منصات مختلفة، من تطبيق الهاتف المحمول إلى واجهة الويب أو حتى برنامج سطر الأوامر command-line باستخدام نفس the model في جميع الحالات، لا علاقة لذلك بكيفية عرض المعلومات للمستخدم.

عندما تفكر في تطبيق الموارد البشرية the HR application من السيناريو الخاص بنا، قد يتضمن the model فئات `Employee` class and `Department` class التي تحتوي على بيانات ومنطق حول هذه الكيانات، قد يتضمن the model أيضًا أشياء مثل `EmployeeRepository` class التي تحتوي على منطق الثبات أو الاستمرارية persistence logic تراعي بعض أنماط تصميم البرامج software-design patterns الأخرى أمورًا مثل مستودعات منفصلة repositories as separate عن the model لكن في سياق the context of MVVM فإننا نشير غالبًا إلى أي منطق أعمال أو بيانات تجارية باعتبارها جزءًا من the model

نعرض فيما يلي مثالين لـ a model في لغة C#

```
public class Employee
{
    public int Id { get; }
    public string Name { get; set; }
    public Employee Supervisor { get; set; }
    public DateTime HireDate { get; set; }
    public void ClockIn() { ... }
}

public class EmployeeRepository
{
   public IList<Employee> QueryEmployees() { ... }
   ...
}
```

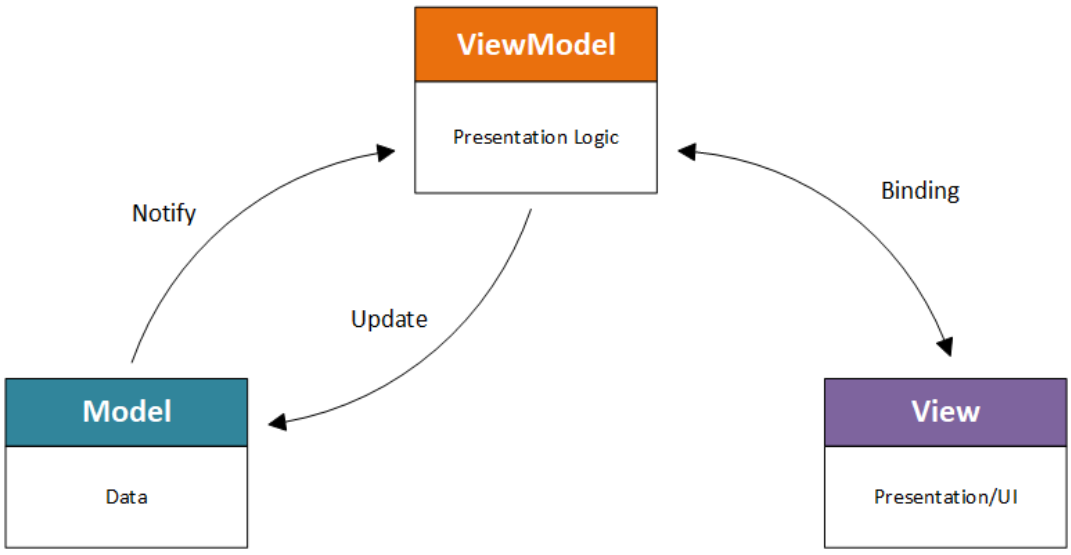
ما هو view

تتحكم التعليمات البرمجية للعرض The view code controls في الأشياء التي تتفاعل مباشرة مع المستخدم، مثل عناصر التحكم مثل الأزرار وحقول الإدخال buttons and entry fields بالإضافة إلى العناصر المرئية البحتة purely visual elements مثل النسق والأنماط والخطوط themes, styles, fonts

في .NET MAUI. لست مضطرًا إلى كتابة أي تعليمات C# لإنشاء view بنفسك، بدلاً من ذلك، غالبًا ما تقوم بتعريف view من خلال ملفات XAML بالطبع، هناك مواقف تتطلب عنصر تحكم مستخدم مخصص، حيث تقوم بإنشاء view الخاصة من خلال التعليمات.

ما هو viewmodel

هذا يعيدنا إلى `viewmodel` هو الوسيط بين business logic (model) and our views (UI)



فكر فيما قد يفعله viewmodel لتطبيق الموارد البشرية، لنفترض أن هناك view يعرض وقت الإجازة المتاح للموظف، وتريد عرض رصيد الإجازة على هيئة "أسبوعين، ٣ أيام، ٤ ساعات" لكن منطق العمل the business logic in the model يوفر نفس القيمة مثل ١٣,٥ يومًا، وهو رقم عشري decimal number يمثل إجمالي الأيام في يوم عمل مدته ٨ ساعات، قد يبدو نموذج الكائن The object model مثل القائمة التالية:

Model – The `Employee class` الذي يتضمن method

```
public decimal GetVacationBalanceInDays()
```

```
{
    //Some math that calculates current vacation
balance
    ...
}
```

ViewModel – class `EmployeeViewModel` يحتوي على خاصية property مثل هذه:

```
public class EmployeeViewModel
{
    private Employee _model;

  public string FormattedVacationBalance
  {
    get
    {
     decimal vacationBalance =
_model.GetVacationBalanceInDays();
     ... // Some logic to format and return the
string as "X weeks, Y days, Z hours"
    }
  }
}
```

View – تحتوي صفحة XAML page على عناصر single label and close button يحتوي label على binding to the viewmodel's property

```
<Label Text="{Binding FormattedVacationBalance}" />
```

كل ما تحتاجه بعد ذلك هو `BindingContext` لصفحة تم تعيينها إلى مثيل instance of `EmployeeViewModel`

في هذا المثال، يحتوي model على منطق العمل business logic هذا المنطق غير مرتبط بجهاز أو جهاز مرئي، يمكنك استخدام نفس المنطق لجهاز محمول أو كمبيوتر سطح مكتب، لا يعرف The view شيئاً عن منطق العمل، تعرف عناصر التحكم في العرض The view controls مثل label كيفية عرض النص على الشاشة، ولكنها لا تهتم إذا كان رصيد عطلة أو سلسلة عشوائية vacation balance or a random string يعرف viewmodel قليلاً من كلاً العالمين، لذا يمكنه العمل كوسيط intermediary

الأمر المثير للاهتمام هو كيف يعمل `viewmodel` كوسيط intermediary فهو يعرض الخصائص التي يمكن أن ترتبط بها `view` الخصائص العامة Public properties هي الطريقة الوحيدة التي يوفر بها `viewmodel` البيانات.

هذا يقودنا إلى سبب تسميته `viewmodel` يمثل "model" في MVVM بنية العمليات التجارية وبياناتها ومنطقها the structure, data, and logic of the business processes ويمثل `viewmodel` البنية والبيانات والمنطق الذي يتطلبه view طريقة العرض the structure, data, and logic that the view requires

كيف تعمل طريقة العرض the view مع the viewmodel

عندما تنظر إلى العلاقة التي تربط viewmodel بالـ the model فهي علاقة قياسية بين فئة وفئة standard class-to-class relationship يحتوي viewmodel على مثيل instance of the model ويعرض جوانب the model لـ view من خلال الخصائص properties ولكن كيف يحصل view على الخصائص ويعينها على the viewmodel عندما يتغير the viewmodel كيف يقوم the view بتحديث عناصر التحكم المرئية بقيم جديدة؟ الإجابة هي ربط البيانات data binding

تتم قراءة خصائص viewmodel في الوقت الذي يكون فيه الكائن مرتبطاً بطرق العرض the object is bound to the view ولكن، الربط ليس لديه طريقة لمعرفة ما إذا كانت خصائص viewmodel تتغير بعد تطبيق الربط على طريقة العرض the view لا يؤدي تغيير viewmodel إلى نشر القيمة الجديدة تلقائيًا من خلال الربط بالـ the view لتحديث viewmodel إلى طريقة العرض، يجب أن ينفذ واجهة `System.ComponentModel.INotifyPropertyChanged`

تعلن واجهة interface `INotifyPropertyChanged` عن حدث واحد يسمى PropertyChanged يأخذ معلمة واحدة parameter هي اسم الخاصية التي غيرت قيمتها، ويفهم نظام الربط المستخدم في .NET MAUI هذه الواجهة ويستمع إلى هذا الحدث، وعندما تتغير خاصية في viewmodel وتثير الحدث، يقوم الربط بإعلام الهدف بالتغيير.

فكر في كيفية عمل ذلك في تطبيق الموارد البشرية باستخدام نموذج عرض الموظف employee viewmodel يحتوي The viewmodel على خصائص تمثل الموظف، مثل اسم الموظف، ينفذ The viewmodel واجهة `INotifyPropertyChanged` وعندما تتغير خاصية الاسم Name property يثير الحدث event `PropertyChanged` على النحو التالي:

```
using System.ComponentModel;

public class EmployeeViewModel : INotifyPropertyChanged
{
  public event PropertyChangedEventHandler? PropertyChanged;
   private Employee _model;

   public string Name
   {
       get {...}
       set
       {
           _model.Name = value;
           OnPropertyChanged(nameof(Name))
       }
   }

   protected void OnPropertyChanged(string
propertyName) =>
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
}
```

يحتوي The view الذي يصف تفاصيل الموظف على عنصر label مرتبط بخاصية الاسم في نموذج العرض the viewmodel's Name property

```
<Label Text="{Binding Name}" />
```

عندما تتغير خاصية الاسم the Name property في the viewmodel يتم رفع حدث PropertyChanged باسم تلك الخاصية. يستمع الربط إلى الحدث ثم يخطر الـ label بأن الخاصية `Name` قد تغيرت، بعد ذلك، يتم تحديث خاصية the label's `Text` بأحدث قيمة.

اختبار بسيط:

١- يعمل أحد المطورين على تطبيق .NET MAUI. ويريد تنفيذ MVVM pattern لقد أنشأ model لبيانات العمل والعمليات business data and operations وطريقة عرض view لتفاعل المستخدم. ما الذي يجب أن ينشئه بعد ذلك ليكون بمثابة وسيط بين the model and the view

٢- يقوم فريق بتطوير تطبيق هواتف باستخدام .NET MAUI. ويريد التأكد من إمكانية نقل منطق العمل الخاص به عبر منصات مختلفة. أين ينبغي لهم وضع منطق العمل هذا في MVVM pattern

حل الاختبار:

١- A viewmodel

في the MVVM pattern يعمل the viewmodel كوسيط بين the model and the view

٢- In the model

في the MVVM pattern يكون the model هو المكان الذي توجد فيه بيانات العمل والعمليات business data and operations

٣ استخدام نموذج عرض viewmodel

بعد التعرف على المكونات التي تشكل نمط that make up the MVVM pattern ربما وجدت أن the model and view كانا سهلي التعريف. دعونا نستكشف كيفية استخدام نموذج العرض the viewmodel لتحديد دوره بشكل أفضل في النمط the pattern

عرض الخصائص لواجهة المستخدم properties to the user interface

كما هو الحال في المثال السابق، تعتمد viewmodels عادةً على models لمعظم بياناتها، وأي منطق تسلسل للأعمال، لكن نموذج العرض viewmodels هو الذي يُنسق البيانات، ويحولها، ويثريها بأي طريقة تتطلبها طريقة العرض view الحالية.

التنسيق باستخدام نموذج عرض Format by using a viewmodel

لقد رأيت بالفعل مثالاً على التنسيق من خلال وقت الإجازة، تنسيق التاريخ، وترميز الأحرف، والتسلسل Date formatting, character encoding, and serialization كلها أمثلة على كيفية تنسيق the viewmodel بيانات من the model

التحويل باستخدام نموذج عرض Convert by using a viewmodel

يوفر أحيانًا the model المعلومات بطرق غير مباشرة، لكن قد تصلح the viewmodel ذلك، على سبيل المثال، افترض أنك تود توضيح ما إذا كان الموظف مشرفًا أم لا، لكن نموذجنا model `Employee` لا يخبرنا بذلك مباشرةً، بدلاً من ذلك، عليك استنتاج هذه الحقيقة بناءً على ما إذا كان الشخص لديه آخرون يقدمون له التقارير. افترض أن the model يتمتع بهذه الخاصية:

```
public IList<Employee> DirectReports
{
    get
    {
        ...
    }
}
```

إذا كانت القائمة فارغة، يمكنك استنتاج أن هذا الموظف `Employee` ليس مشرفًا، في هذه الحالة، يتضمن `EmployeeViewModel` الخاصية `IsSupervisor` التي توفر هذا المنطق:

```
public bool IsSupervisor => _model.DirectReports.Any();
```

الإثراء باستخدام نموذج عرض Enrich by using a viewmodel

في بعض الأحيان قد يوفر النموذج model معرّفًا للبيانات ذات الصلة فقط ID for related data أو قد تحتاج إلى الانتقال إلى عدة فئات نماذج model classes لربط البيانات المطلوبة لشاشة واحدة، يوفر نموذج العرض The viewmodel مكانًا مثاليًا لأداء هذه المهام أيضًا، افترض أنك تريد إظهار كافة المشاريع التي يديرها موظف حاليًا، تلك البيانات ليست جزءًا من model class `Employee` يمكن الوصول إليها من خلال النظر إلى فئة نموذج model class `CompanyProjects` يعرض نموذج عرض الموظف EmployeeViewModel كما هو الحال دائمًا، عمله كخاصية عامة:

```
public IEnumerable<string> ActiveProjects =>
CompanyProjects.All
    .Where(p => p.Owner == _model.Id && p.IsActive)
    .Select(p => p.Name);
```

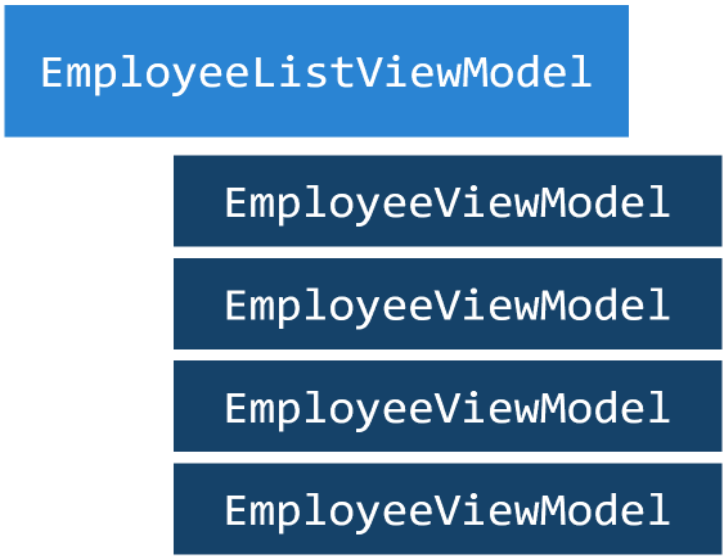
استخدام خصائص التمرير pass-through properties مع نموذج العرض viewmodel

يحتاج viewmodel في كثير من الأحيان إلى الخاصية التي يوفرها model فيما يتعلق بهذه الخصائص، يعرض viewmodel البيانات فقط من خلال:

```
public string Name
{
    get => _model.Name;
    set => _model.Name = value;
}
```

تعيين نطاق Set the scope نموذج العرض the viewmodel

يمكنك استخدام نموذج عرض viewmodel في أي مستوى يوجد فيه طريقة عرض view تحتوي الصفحة عادةً على viewmodel ولكن قد يكون ذلك من خلال العروض الفرعية للصفحة ة أحد الأسباب الشائعة لنماذج العرض المتداخلة nested viewmodels هو عندما تعرض الصفحة ListView على الصفحة. تحتوي هذه القائمة The list على viewmodel يمثل المجموعة، مثل `EmployeeListViewModel` كل عنصر في القائمة the list هو `EmployeeViewModel`



من الشائع أيضًا أن يكون لديك نموذج عرض على المستوى الأعلى top-level viewmodel يحتفظ بالبيانات والحالة للتطبيق بأكمله، ولكنه غير مرتبط بأي صفحة معينة، يُستخدم نموذج العرض viewmodel عادةً للحفاظ على العنصر النشط the "active" item ضع في اعتبارك المثال الموضح ListView للتو، عندما يحدد المستخدم سجل موظف employee row يمثل هذا الموظف العنصر الحالي، إذا انتقل المستخدم إلى صفحة التفاصيل detail page أو حدد زر شريط أدوات toolbar button أثناء تحديد هذا الصف، يجب أن يكون الإجراء أو العرض مخصصًا لهذا الموظف، تتمثل إحدى الطرق الأنيقة للتعامل مع هذا السيناريو في ربط بيانات data-bound بخاصية يمكن لشريط `ListView.SelectItem` الأدوات أو صفحة التفاصيل toolbar or detail page الوصول إليها أيضًا. يعمل وضع هذه الخاصية على نموذج عرض مركزي central viewmodel بشكل جيد.

تحديد الوقت المناسب لإعادة استخدام viewmodels مع views

تعتمد كيفية تحديد العلاقة بين viewmodel and model وبين viewmodel and view بشكل كبير من خلال متطلبات التطبيق أكثر من القواعد، يتمثل الغرض من viewmodel في توفير البنية والبيانات structure and data التي يحتاجها view ينبغي أن توجه القرارات بشأن مدى حجم "how big" نطاق عرض النموذج scope a viewmodel

تعكس نماذج العروض Viewmodels غالبًا بنية فئة النموذج structure of a model class ولديها علاقة فردية مع تلك الفئة one-to-one relationship with that class رأيت مثالاً سابقًا مع `EmployeeViewModel` المُضمنة، الذي قام بتغليف وتعزيز مثيل واحد wrapped and augmented a single `Employee` instance لكنها ليست دائمًا علاقة فردية، إذا تم تصميم

viewmodel لتوفير ما تحتاجه view فقد ينتهي بك الأمر بدلاً من ذلك إلى شيء مثل `HRDashboardViewModel` لتقديم نظرة عامة على قسم الموارد البشرية، الذي ليس له علاقة صريحة بأي نموذج model ولكن يمكنه استخدام البيانات من أي فئة نموذج model class

على نحو مماثل، قد تجد أن viewmodels and views غالبًا ما تكون لها علاقة فردية one-to-one relationship ولكن هذا ليس بالضرورة الحالة الفعلية، دعنا نفكر مرة أخرى في ListView الذي يعرض سجلاً لكل موظف، عند تحديد أحد الصفوف، تنتقل إلى صفحة تفاصيل الموظف employee-detail page

تحتوي صفحة القائمة على viewmodel خاص بها مع مجموعة، وكما اقترحنا سابقًا، يمكن أن تكون هذه المجموعة عبارة عن مجموعة من كائنات objects `EmployeeViewModel` وعندما يختار المستخدم سجلاً، يمكن تمرير مثيل `EmployeeViewModel` إلى `EmployeeDetailPage` ويمكن لصفحة التفاصيل استخدام ذلك `EmployeeViewModel` باعتباره `BindingContext` الخاص بها.

قد يكون هذا السيناريو فرصة ممتازة لإعادة استخدام viewmodel ولكن عليك مراعاة أن viewmodel تهدف إلى توفير ما يحتاجه view في بعض الحالات، قد ترغب في وجود نماذج عرض منفصلة separate viewmodels حتى لو كانت جميعها تستند إلى نفس model class في هذا المثال، من المرجح أن تحتاج صفوف ListView إلى معلومات أقل بكثير من صفحة التفاصيل الكاملة، إذا كان استرداد البيانات التي تحتاجها صفحة التفاصيل يضيف الكثير من الحمل الزائد a lot if overhead فقد ترغب في الحصول على كل من `EmployeeListRowViewModel`, `EmployeeDetailViewModel` models اللذين يخدمان views المعنية هذه.

نموذج كائن Viewmodel object model

استخدام فئة أساسية تنفذ `INotifyPropertyChanged` يعني أنك لست بحاجة إلى إعادة تنفيذ الواجهة على كل viewmodel فكر في تطبيق الموارد البشرية كما هو موضح في الجزء السابق من وحدة التدريب هذه، نفذت فئة class `EmployeeViewModel` وواجهة `INotifyPropertyChanged` interface ووفرت أسلوب مساعد helper method يسمى `OnPropertyChanged` لرفع حدث `PropertyChanged` event تتطلب viewmodels الأخرى في المشروع، مثل تلك التي تصف الموارد المعينة لموظف، سوف يتطلب أيضًا `INotifyPropertyChanged` التكامل الكامل مع view

مكتبة أدوات The MVVM Toolkit library وهي جزء من أدوات مجتمع .NET Community Toolkit عبارة عن مجموعة من الأنواع القياسية، الخفيفة الوزن، والمستقلة التي توفر تنفيذًا أوليًا لبناء تطبيقات حديثة باستخدام نمط the MVVM pattern

بدلاً من كتابة فئة أساسية viewmodel base class خاصة بك، يمكنك الوراثة من الفئة الموجودة في مجموعة الأدوات toolkit's `ObservableObject` class والتي توفر كل ما تحتاجه لفئة viewmodel base class يمكن تبسيط `EmployeeViewModel` من:

```
using System.ComponentModel;

public class EmployeeViewModel : INotifyPropertyChanged
{
public event PropertyChangedEventHandler? PropertyChanged;
private Employee _model;

    public string Name
    {
        get {...}
        set
        {
            _model.Name = value;
            OnPropertyChanged(nameof(Name))
        }
    }

    protected void OnPropertyChanged(string
propertyName) =>
         PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
}
```

إلى التعليمة البرمجية التالية:

```
using Microsoft.Toolkit.Mvvm.ComponentModel;

public class EmployeeViewModel : ObservableObject
{
    private string _name;
```

```
    public string Name
    {
        get => _name;
        set => SetProperty(ref _name, value);
    }
}
```

يمكن تبسيط التعليمات البرمجية بشكل أكبر باستخدام مولدات المصدر source generators التي توفرها مجموعة أدوات the MVVM Toolkit من خلال إنشاء الفئة `partial` the class وإضافة `[ObservableProperty]` إلى المتغير الخاص the private variable يتم إنشاء الخاصية Name العامة مع إشعارات تغيير الخاصية المناسبة.

```
using Microsoft.Toolkit.Mvvm.ComponentModel;

public partial class EmployeeViewModel : ObservableObject
{
    [ObservableProperty]
    private string _name;
}
```

يتم توزيع مجموعة أدوات The MVVM Toolkit من خلال حزمة the `CommunityToolkit.Mvvm` NuGet package

اختبار بسيط:

١- عند استخدام نمط MVVM pattern مع .NET MAUI. لا يتم فصل model, view, and viewmodel بشكل كامل عن بعضهم البعض. أيّ من الخيارات يصف تبعية مشتركة واحدة بين أجزاء MVVM

٢- أيّ مما يلي من المُرجح أن يكون مرتبطًا ارتباطًا وثيقًا بنظام التشغيل، ويصعب إنشاء unit tests اختبارات وحدة له: the model, the view, the viewmodel

حل الاختبار:

١- يحتوي نموذج العرض viewmodel على معلومات مباشرة خاصة بالنموذج model

يمكن لنموذج العرض viewmodel أن يصل إلى الخصائص والأساليب properties and methods على النموذج model مباشرةً

٢- طرق العرض The view

في .NET MAUI. يتم تنفيذ view بشكل أساسي بواسطة XAML ولكن من الصعب إجراء اختبار الوحدة، حتى إذا تم ترميزه.

٤ الملخص

في هذه الوحدة، تعرفت على نمط Model-View-ViewModel (MVVM) وتطبيقه في تطبيقات واجهة مستخدم التطبيقات متعددة الأنظمة .NET Multi-platform App UI (MAUI) apps The MVVM pattern يساعد نمط في نقل معظم منطق سلوك واجهة المستخدم إلى فئات قابلة للاختبار للوحدة unit-testable classes تسمى نماذج العرض viewmodels وبالتالي معالجة تحدي كتابة اختبارات الوحدة مقابل ملفات التعليمات البرمجية الخلفية code-behind كما تعرفت على أدوار model, view, viewmodel في تطبيق MVVM application تتعمق الوحدة في مسؤوليات نموذج العرض viewmodel بما في ذلك تنسيق البيانات وتحويلها وإثرائها formatting, converting, and enriching data من النموذج the model لتناسب متطلبات العرض الحالي. كما تم تقديم مكتبة مجموعة ادوات MVVM Toolkit التي توفر تنفيذًا أوليًا لبناء تطبيقات حديثة باستخدام نمط MVVM

تتضمن النقاط الرئيسية المستفادة من هذه الوحدة، فهم دور نموذج العرض the viewmodel وكيفية تصميم نموذج العرض ليناسب طريقة العرض view وشمل ذلك تحديث MovieListViewModel لإضافة خاصية SelectedMovie property وتحسين التحديد في طريقة عرض MovieListPage view باستخدام نمط MVVM بالإضافة إلى ذلك، تعلمت عن نمط الأوامر the command pattern كحل للتعامل مع مكونات واجهة المستخدم مثل تنشيط الزر أو عنصر القائمة Button or MenuItem activation