le Aplicaciones Multiplataforma con

Material Completo del Curso

Total de Unidades: 5

31 de December de 2025

Índice de Contenidos

Desarrollo de Aplicaciones Multiplataforma con .NET MAUI

Unidad 01: Introducción a .NET MAUI

Notas del Presentador:
Bienvenidos al curso de .NET MAUI. Esta primera unidad establece los fundamentos y el contexto histórico. Es crucial que los estudiantes comprendan la evolución desde Xamarin.Forms y las ventajas de .NET MAUI.

Competencias y Resultados de Aprendizaje

Competencia Fundamental:

CF4: Aplicar de forma intensiva y crítica las tecnologías de la información y la comunicación en las actividades profesionales.

Resultados de Aprendizaje:

- Comprender qué es .NET MAUI y su propósito
- Conocer la evolución histórica desde Xamarin.Forms
- Identificar las ventajas del desarrollo multiplataforma
- Entender la arquitectura de .NET MAUI
- Configurar el entorno de desarrollo correctamente
- Crear y ejecutar una primera aplicación MAUI
- Reconocer la estructura de un proyecto MAUI

Notas del Presentador:
Estos resultados están alineados con el programa académico. Asegúrate de que los estudiantes comprendan que .NET MAUI es una tecnología moderna y actualizada.

Agenda de la Unidad

1. ¿Qué es .NET MAUI?
2. Historia y evolución (Xamarin.Forms → .NET MAUI)
3. Ventajas de desarrollo multiplataforma
4. Arquitectura de .NET MAUI
5. Requisitos del sistema
6. Instalación y configuración del entorno
7. Visual Studio 2022 y .NET MAUI
8. Crear tu primera aplicación MAUI
9. Estructura de un proyecto MAUI
10. Práctica guiada
11. Resumen y evaluación

Notas del Presentador:
Duración estimada: 4 horas. Distribución: Teoría (2h), Instalación y Configuración (1h), Práctica (0.5h), Evaluación (0.5h).

¿Qué es .NET MAUI?

Definición: .NET Multi-platform App UI (.NET MAUI) es un framework multiplataforma para crear aplicaciones nativas móviles y de escritorio con C# y XAML.

Características Principales:

- **Multiplataforma:** Una sola base de código para Android, iOS, macOS, Windows y más
- **Nativo:** Compila a código nativo de cada plataforma
- **Moderno:** Basado en .NET 6+ con mejor rendimiento
- **XAML:** Interfaz declarativa para diseñar UI
- **C#:** Lógica de negocio con un lenguaje moderno
- **Hot Reload:** Ver cambios en tiempo real

Plataformas Soportadas: Android, iOS, macOS, Windows (WinUI 3), Tizen (opcional)

Historia y Evolución

Línea de Tiempo:

- **2011:** Xamarin se funda, permitiendo desarrollo móvil multiplataforma con C#
- **2014:** Microsoft adquiere Xamarin
- **2016:** Xamarin.Forms se convierte en open source
- **2019:** Microsoft anuncia .NET MAUI como sucesor de Xamarin.Forms
- **2020:** .NET 5 lanzado, preparando el camino para MAUI
- **2022:** .NET MAUI lanzado oficialmente con .NET 6
- **2023+:** Mejoras continuas con .NET 7, 8 y futuras versiones

> **Migración:** Xamarin.Forms sigue siendo soportado, pero .NET MAUI es el futuro. Microsoft recomienda migrar a MAUI para nuevos proyectos.

Xamarin.Forms vs .NET MAUI

Principales Diferencias:

- **Base Tecnológica:**
    - Xamarin.Forms: Basado en .NET Standard
    - .NET MAUI: Basado en .NET 6+ (más moderno y rápido)
- **Plataformas:**
    - Xamarin.Forms: Android, iOS, UWP, WPF
    - .NET MAUI: Android, iOS, macOS, Windows (WinUI 3), Tizen
- **Rendimiento:**
    - MAUI tiene mejor rendimiento y menor tamaño de aplicación
- **Arquitectura:**
    - MAUI tiene una arquitectura más limpia y modular

Ventajas de Desarrollo Multiplataforma

Eficiencia y Productividad:

- **Una sola base de código:** Escribe una vez, ejecuta en todas las plataformas
- **Menor tiempo de desarrollo:** No necesitas equipos separados para cada plataforma
- **Mantenimiento simplificado:** Un solo código base para actualizar
- **Hot Reload:** Ver cambios instantáneamente sin recompilar

Costos y Recursos:

- **Reducción de costos:** Menos desarrolladores necesarios
- **Reutilización de código:** Hasta 90% de código compartido
- **Conocimientos existentes:** Si sabes C# y XAML, puedes desarrollar para todas las plataformas

Calidad y Consistencia:

- **Experiencia consistente:** Misma UI/UX en todas las plataformas
- **Rendimiento nativo:** Compila a código nativo
- **Acceso a APIs nativas:** Puedes usar funcionalidades específicas de cada plataforma

Arquitectura de .NET MAUI

Arquitectura en Capas:

Componentes Clave:

- **Handlers:** Traducen controles MAUI a controles nativos
- **Renderers:** Convierten XAML en UI nativa
- **Plataforma Específica:** Código específico por plataforma cuando es necesario

Requisitos del Sistema

Para Desarrollo en Windows:

- **SO:** Windows 10 versión 1903 o superior / Windows 11
- **Visual Studio 2022:** Community, Professional o Enterprise
- **Workloads:** Desarrollo para .NET Multiplataforma
- **.NET SDK:** .NET 6.0 o superior
- **Android:** Android SDK (incluido en Visual Studio)
- **Windows:** Windows SDK (incluido en Visual Studio)

Para Desarrollo en macOS:

- **SO:** macOS 10.15 o superior
- **Visual Studio para Mac** o **Visual Studio Code**
- **Xcode:** Para desarrollo iOS/macOS
- **.NET SDK:** .NET 6.0 o superior

> **Nota:** Para desarrollar para iOS necesitas macOS, incluso si desarrollas en Windows (puedes usar un Mac remoto).

Instalación y Configuración del Entorno

Paso 1: Instalar Visual Studio 2022

1. Descargar Visual Studio 2022 desde visualstudio.microsoft.com
2. Ejecutar el instalador
3. Seleccionar workload: **"Desarrollo para .NET Multiplataforma"**
4. Incluir componentes:
    - .NET Multi-platform App UI development
    - Android SDK setup
    - Windows SDK

Paso 2: Verificar Instalación

```
dotnet --version
dotnet workload list
```

Paso 3: Instalar Workload de MAUI

```
dotnet workload install maui
```

Visual Studio 2022 y .NET MAUI

Características de Visual Studio para MAUI:

- **Diseñador XAML:** Editor visual para crear interfaces
- **Hot Reload:** Ver cambios en tiempo real
- **Emuladores Integrados:** Android Emulator, iOS Simulator
- **Depuración:** Depurar en múltiples plataformas
- **IntelliSense:** Autocompletado para XAML y C#
- **Plantillas de Proyecto:** Plantillas listas para usar

Extensiones Útiles:

- **.NET MAUI Check:** Verifica que todo esté configurado correctamente
- **XAML Styler:** Formatea automáticamente XAML
- **LiveXAML:** Preview en tiempo real de XAML

Crear tu Primera Aplicación MAUI

Método 1: Desde Visual Studio

1. Abrir Visual Studio 2022
2. Crear nuevo proyecto
3. Buscar: **"Aplicación .NET MAUI"**
4. Configurar nombre y ubicación
5. Seleccionar .NET 6.0 o superior
6. Click en "Crear"

Método 2: Desde Línea de Comandos

```
dotnet new maui -n MiPrimeraApp
cd MiPrimeraApp
dotnet build
dotnet run
```

> **Consejo:** La primera compilación puede tardar varios minutos mientras se descargan dependencias.

Estructura de un Proyecto MAUI

```
MiPrimeraApp/
├── Platforms/
│   ├── Android/
│   │   ├── MainActivity.cs
│   │   └── AndroidManifest.xml
│   ├── iOS/
│   │   ├── AppDelegate.cs
│   │   └── Info.plist
│   ├── Windows/
│   │   └── App.xaml
│   └── MacCatalyst/
│       └── AppDelegate.cs
├── Resources/
│   ├── Images/
│   ├── Fonts/
│   └── Styles/
│       └── Styles.xaml
├── App.xaml
├── App.xaml.cs
├── AppShell.xaml
├── MainPage.xaml
├── MainPage.xaml.cs
└── MauiProgram.cs
```

Archivos Clave:

- **MauiProgram.cs:** Configuración de la aplicación
- **App.xaml:** Recursos globales y configuración
- **AppShell.xaml:** Navegación principal (Shell)
- **MainPage.xaml:** Página principal de la aplicación
- **Platforms/:** Código específico por plataforma

MauiProgram.cs - Configuración

```
using Microsoft.Maui;
using Microsoft.Maui.Hosting;
using Microsoft.Extensions.Logging;

namespace MiPrimeraApp;

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

        #if DEBUG
        builder.Logging.AddDebug();
        #endif

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

¿Qué hace este archivo?

- Configura la aplicación MAUI
- Registra fuentes personalizadas
- Configura servicios y dependencias
- Es el punto de entrada de configuración

MainPage.xaml - Interfaz de Usuario

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

    <ScrollView>
        <VerticalStackLayout
            Spacing="25"
            Padding="30,0"
            VerticalOptions="Center">

            <Image Source="dotnet_bot.png"
                   HeightRequest="200"
                   HorizontalOptions="Center" />

            <Label Text="Bienvenido a .NET MAUI"
                   FontSize="32"
                   HorizontalOptions="Center" />

            <Button Text="Haz clic aquí"
                    Clicked="OnCounterClicked"
                    HorizontalOptions="Center" />
        </VerticalStackLayout>
    </ScrollView>
</ContentPage>
```

MainPage.xaml.cs - Lógica

```
namespace MiPrimeraApp;

public partial class MainPage : ContentPage
{
    int count = 0;

    public MainPage()
    {
        InitializeComponent();
    }

    private void OnCounterClicked(object sender, EventArgs e)
    {
        count++;

        if (count == 1)
            CounterBtn.Text = $"Haz clic {count} vez";
        else
            CounterBtn.Text = $"Haz clic {count} veces";

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

Conceptos Clave:

- **partial class:** Parte del código generado por XAML
- **InitializeComponent():** Carga el XAML
- **Event Handlers:** Métodos que responden a eventos

Práctica Guiada #1: Tu Primera App MAUI

Objetivo:

Crear y ejecutar tu primera aplicación .NET MAUI funcional.

Paso 1: Crear el Proyecto

1. Abre Visual Studio 2022
2. Crea un nuevo proyecto "Aplicación .NET MAUI"
3. Nómbralo: **MiPrimeraAppMAUI**
4. Selecciona .NET 6.0 o superior

Paso 2: Explorar la Estructura

1. Examina la carpeta Platforms
2. Revisa App.xaml y MauiProgram.cs
3. Abre MainPage.xaml y MainPage.xaml.cs

Paso 3: Modificar la Interfaz

1. Cambia el texto del Label a "Hola, .NET MAUI!"
2. Agrega un Entry (campo de texto) debajo del Label
3. Agrega un segundo Button que muestre el texto del Entry

Paso 4: Compilar y Ejecutar

1. Selecciona una plataforma objetivo (Android, Windows, etc.)

2. Presiona F5 para compilar y ejecutar

3. Verifica que la aplicación funcione correctamente

Mini-Quiz

Pregunta 1:

¿Qué significa MAUI?

Respuesta: Multi-platform App UI - Interfaz de Aplicación Multiplataforma

Pregunta 2:

¿Cuál es la principal ventaja de usar .NET MAUI sobre desarrollo nativo separado?

Respuesta: Una sola base de código para múltiples plataformas, reduciendo tiempo de desarrollo y mantenimiento.

Pregunta 3:

¿Qué archivo es responsable de la configuración inicial de una aplicación MAUI?

Respuesta: MauiProgram.cs - contiene el método CreateMauiApp() que configura la aplicación.

Resumen de la Unidad

Puntos Clave Aprendidos:

- .NET MAUI es un framework multiplataforma moderno para crear apps nativas
- Evolucionó desde Xamarin.Forms, mejorando rendimiento y arquitectura
- Permite desarrollar para Android, iOS, macOS, Windows con una sola base de código
- Visual Studio 2022 es la herramienta principal de desarrollo
- La estructura de proyecto incluye código compartido y código específico por plataforma
- XAML se usa para UI y C# para lógica de negocio
- Hot Reload permite ver cambios en tiempo real

Conexión con la Siguiente Unidad:

En la Unidad 2 aprenderemos los fundamentos de XAML y los diferentes layouts disponibles para crear interfaces de usuario atractivas y responsivas.

Tarea para Casa

Actividades a Realizar:

1. **Instalación y Configuración (30 puntos):**
    - Instalar Visual Studio 2022 con workload de .NET MAUI
    - Verificar instalación ejecutando: `dotnet workload list`
    - Capturar pantalla de la verificación exitosa
2. **Primera Aplicación (40 puntos):**
    - Crear una aplicación MAUI llamada "MiPerfilApp"
    - Agregar un Label con tu nombre
    - Agregar un Label con tu carrera
    - Agregar un Button que muestre un mensaje de saludo
    - Ejecutar la app en al menos una plataforma
3. **Investigación (30 puntos):**
    - Investigar 3 aplicaciones reales desarrolladas con .NET MAUI
    - Crear un documento de 1-2 páginas describiendo cada app
    - Incluir capturas de pantalla si es posible

Entregables:

- Código fuente del proyecto "MiPerfilApp" (carpeta completa)
- Documento Word/PDF con investigación y capturas
- Formato: Arial 12, márgenes normales, portada con datos del estudiante

Fecha de Entrega:

Una semana después de esta clase (fecha a definir por el instructor)

Criterios de Evaluación:

- Instalación correcta del entorno (30%)
- Funcionalidad de la aplicación (40%)
- Calidad de la investigación (20%)
- Formato y presentación (10%)

Recursos Adicionales

📚 Material Complementario

Para ampliar tu conocimiento sobre esta unidad, visita nuestra página de recursos con:

- Documentación oficial de Microsoft sobre .NET MAUI
- Videos tutoriales de introducción
- Guías de instalación paso a paso
- Ejemplos de código y repositorios
- Comunidad y foros de .NET MAUI

Ver Recursos Adicionales →

Recursos Adicionales - Unidad 01

Introducción a .NET MAUI - Enlaces, videos y material complementario

📚 Documentación Oficial de Microsoft

[Documentación Oficial .NET MAUI Oficial]

Documentación completa y actualizada de .NET MAUI directamente de Microsoft. Incluye guías, tutoriales, ejemplos y referencias de API.

[Guía de Inicio Rápido Tutorial]

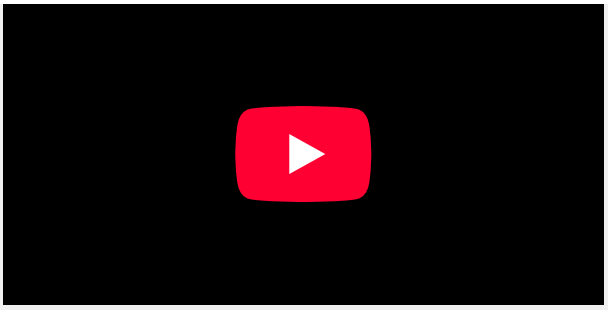
Paso a paso para crear tu primera aplicación .NET MAUI. Ideal para empezar desde cero.

[Guía de Instalación Instalación]

Instrucciones detalladas para instalar y configurar .NET MAUI en Windows y macOS.

📹 Videos Educativos

Introducción a .NET MAUI



Video oficial de Microsoft introduciendo .NET MAUI y sus características principales.

[.NET MAUI: ¿Qué es y por qué usarlo? Video]

Explicación clara sobre qué es .NET MAUI y las ventajas de usarlo para desarrollo multiplataforma.

[Instalación y Primer Proyecto Tutorial]

Tutorial paso a paso sobre cómo instalar Visual Studio 2022 y crear tu primer proyecto MAUI.

🛠️ Herramientas Recomendadas

[Visual Studio 2022 Gratis]

IDE principal para desarrollo .NET MAUI. Incluye diseñador XAML, depuración multiplataforma y Hot Reload.

[.NET MAUI Check Tool Utilidad]

Herramienta de línea de comandos que verifica que tu entorno esté correctamente configurado para .NET MAUI.

XAML Styler Extensión

Extensión para Visual Studio que formatea automáticamente archivos XAML según mejores prácticas.

📖 Libros y Tutoriales

.NET MAUI Cross-Platform Application Development - Packt

Libro completo sobre desarrollo con .NET MAUI, cubriendo desde conceptos básicos hasta temas avanzados.

¿Qué es .NET MAUI? - Microsoft Learn

Artículo oficial explicando en detalle qué es .NET MAUI, su arquitectura y beneficios.

💡 Artículos Especializados

Introduciendo .NET MAUI - Blog de Microsoft

Artículo del blog oficial de .NET anunciando .NET MAUI y sus características principales.

Reddit - r/dotnetMAUI

Comunidad activa de desarrolladores .NET MAUI compartiendo experiencias, problemas y soluciones.

📦 Ejemplos de Código y Repositorios

[.NET MAUI Samples GitHub]

Repositorio oficial de Microsoft con ejemplos de código .NET MAUI. Incluye muestras de controles, layouts, navegación, etc.

[Repositorio .NET MAUI GitHub]

Código fuente de .NET MAUI en GitHub. Útil para entender la implementación interna y reportar issues.

[eShopOnContainers Ejemplo]

Aplicación de ejemplo completa mostrando arquitectura MVVM, servicios, y mejores prácticas.

🎓 Cursos Online

[Microsoft Learn - Ruta de Aprendizaje .NET MAUI]

Ruta de aprendizaje oficial gratuita de Microsoft con módulos interactivos sobre .NET MAUI.

> [Pluralsight - Cursos .NET MAUI]
>
> Cursos profesionales sobre .NET MAUI (requiere suscripción de pago).

> ### 💡 Consejo de Estudio
>
> Para esta unidad introductoria, te recomendamos: 1) Ver los videos introductorios para entender el contexto, 2) Seguir la guía oficial de instalación paso a paso, 3) Crear tu primer proyecto y experimentar, 4) Explorar los ejemplos en GitHub para ver código real. La práctica es esencial desde el inicio.

Desarrollo de Aplicaciones Multiplataforma con .NET MAUI

Unidad 02: Fundamentos de XAML y Layouts

Notas del Presentador:
Esta unidad es fundamental para entender cómo crear interfaces en MAUI. XAML es el lenguaje declarativo que permite diseñar UI de forma visual y estructurada.

Competencias y Resultados de Aprendizaje

Resultados de Aprendizaje:

- Comprender qué es XAML y su sintaxis básica
- Conocer los controles básicos de .NET MAUI
- Dominar los diferentes layouts disponibles
- Aplicar propiedades y bindings básicos
- Crear estilos y recursos reutilizables
- Diseñar interfaces responsivas

Agenda de la Unidad

1. Introducción a XAML
2. Sintaxis básica de XAML
3. Controles básicos (Label, Button, Entry, etc.)
4. Layouts principales: StackLayout, Grid, FlexLayout
5. AbsoluteLayout y RelativeLayout
6. Propiedades y bindings básicos
7. Estilos y recursos
8. Responsive design en MAUI
9. Práctica guiada

¿Qué es XAML?

XAML (eXtensible Application Markup Language): Es un lenguaje declarativo basado en XML que permite definir interfaces de usuario de forma estructurada.

Características:

- **Declarativo:** Describe QUÉ quieres, no CÓMO hacerlo
- **Separación de UI y Lógica:** XAML para diseño, C# para código
- **Visual:** Fácil de leer y entender
- **Herencia:** Los elementos pueden heredar propiedades
- **Data Binding:** Conecta UI con datos

Ventaja: XAML permite diseñar interfaces complejas de forma más clara que código C# puro.

Sintaxis Básica de XAML

Estructura de un Archivo XAML:

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

    <Label Text="Hola Mundo" />
</ContentPage>
```

Elementos Clave:

- **xmlns:** Namespace de .NET MAUI
- **xmlns:x:** Namespace para atributos XAML (x:Class, x:Name)
- **x:Class:** Conecta XAML con la clase C# code-behind
- **Elementos:** Representan controles (Label, Button, etc.)
- **Atributos:** Propiedades de los controles

Controles Básicos - Label

```
<Label Text="Mi Texto"
       FontSize="18"
       TextColor="Blue"
       HorizontalOptions="Center"
       VerticalOptions="Center" />
```

Propiedades Comunes:

- **Text:** Contenido del texto
- **FontSize:** Tamaño de fuente
- **TextColor:** Color del texto
- **HorizontalOptions/VerticalOptions:** Alineación (Start, Center, End, Fill)
- **FontAttributes:** Bold, Italic, None

Controles Básicos - Button y Entry

Button:

```
<Button Text="Haz Clic"
        Clicked="OnButtonClicked"
        BackgroundColor="Green"
        TextColor="White"
        CornerRadius="10" />
```

Entry (Campo de Texto):

```
<Entry Placeholder="Escribe aquí"
       Text="{Binding Nombre}"
       Keyboard="Email"
       IsPassword="False" />
```

StackLayout - Layout Vertical y Horizontal

> **StackLayout:** Organiza elementos en una pila, ya sea vertical u horizontalmente.

VerticalStackLayout (Recomendado en MAUI):

```
<VerticalStackLayout Spacing="10" Padding="20">
    <Label Text="Título" />
    <Entry Placeholder="Nombre" />
    <Button Text="Enviar" />
</VerticalStackLayout>
```

Propiedades:

- **Spacing:** Espacio entre elementos
- **Padding:** Espacio interno del layout
- **Orientation:** Vertical (default) u Horizontal

Grid - Layout en Cuadrícula

> **Grid:** Organiza elementos en filas y columnas, ideal para diseños complejos.

```
<Grid RowDefinitions="Auto, *, Auto"
      ColumnDefinitions="*, *"
      RowSpacing="10"
      ColumnSpacing="10">

    <Label Text="Título"
           Grid.Row="0" Grid.ColumnSpan="2" />
    <Entry Grid.Row="1" Grid.Column="0" />
    <Entry Grid.Row="1" Grid.Column="1" />
    <Button Text="OK"
            Grid.Row="2" Grid.ColumnSpan="2" />
</Grid>
```

Conceptos:

- **RowDefinitions/ColumnDefinitions:** Define tamaño de filas/columnas (Auto, *, número)
- **Grid.Row/Grid.Column:** Posición del elemento
- **Grid.ColumnSpan:** Elemento ocupa múltiples columnas

FlexLayout - Layout Flexible

> **FlexLayout:** Layout moderno basado en CSS Flexbox, ideal para diseños adaptativos.

```
<FlexLayout Direction="Row"
            JustifyContent="SpaceAround"
            AlignItems="Center"
            Wrap="Wrap">

    <Button Text="Btn 1" />
    <Button Text="Btn 2" />
    <Button Text="Btn 3" />
</FlexLayout>
```

Propiedades Clave:

- **Direction:** Row o Column
- **JustifyContent:** Start, Center, End, SpaceAround, SpaceBetween
- **AlignItems:** Start, Center, End, Stretch
- **Wrap:** NoWrap o Wrap

AbsoluteLayout y RelativeLayout

AbsoluteLayout:

Posiciona elementos en coordenadas absolutas (x, y).

```
<AbsoluteLayout>
    <Label Text="Título"
           AbsoluteLayout.LayoutBounds="0,0,200,50"
           AbsoluteLayout.LayoutFlags="None" />
</AbsoluteLayout>
```

RelativeLayout:

Posiciona elementos relativos a otros elementos o al contenedor.

> **Nota:** AbsoluteLayout y RelativeLayout son menos comunes en MAUI moderno. Prefiere Grid o FlexLayout para mejor rendimiento.

Propiedades y Bindings Básicos

Propiedades en XAML:

```
<Label Text="Hola"
       FontSize="{StaticResource TamañoFuente}"
       TextColor="{Binding ColorTexto}"
       IsVisible="{Binding EsVisible}" />
```

Tipos de Binding:

- **StaticResource:** Referencia a recurso estático
- **Binding:** Enlace a datos (MVVM)
- **x:Static:** Referencia a propiedad estática
- **Valor Directo:** Text="Hola" (hardcoded)

Estilos y Recursos

Definir Estilos:

```
<ContentPage.Resources>
    <Style x:Key="TituloEstilo" TargetType="Label">
        <Setter Property="FontSize" Value="24" />
        <Setter Property="TextColor" Value="Blue" />
        <Setter Property="FontAttributes" Value="Bold" />
    </Style>
</ContentPage.Resources>

<Label Text="Mi Título" Style="{StaticResource TituloEstilo}" />
```

Colores y Recursos:

```
<Color x:Key="ColorPrincipal">#4CAF50</Color>
<x:Double x:Key="TamañoFuente">18</x:Double>
```

Responsive Design en MAUI

Estrategias:

- **Layouts Flexibles:** Usa Grid con * y Auto
- **OnPlatform:** Valores diferentes por plataforma
- **DeviceInfo:** Detectar tamaño de pantalla
- **Adaptive Triggers:** Cambiar layout según tamaño

Ejemplo OnPlatform:

```
<Label FontSize="{OnPlatform Android=16, iOS=18, WinUI=20}" />
```

Ejemplo DeviceInfo:

```
if (DeviceInfo.Idiom == DeviceIdiom.Phone) {
    // Layout para móvil
} else {
    // Layout para tablet
}
```

Práctica Guiada #2: Crear Formulario con Layouts

Objetivo:

Crear un formulario de registro usando diferentes layouts.

Pasos:

1. Crea un nuevo proyecto MAUI
2. Diseña un formulario con:
    - Grid para organizar campos (2 columnas)
    - Labels y Entries para: Nombre, Email, Teléfono
    - VerticalStackLayout para botones
    - Estilos para los Labels
3. Agrega validación básica
4. Prueba en diferentes plataformas

Mini-Quiz

Pregunta 1:

¿Cuál es la diferencia entre VerticalStackLayout y Grid?

Respuesta: VerticalStackLayout organiza elementos en una pila vertical, mientras que Grid permite organizar elementos en filas y columnas para diseños más complejos.

Pregunta 2:

¿Qué significa Grid.ColumnSpan?

Respuesta: Permite que un elemento ocupe múltiples columnas en un Grid.

Resumen de la Unidad

Puntos Clave:

- XAML es un lenguaje declarativo para diseñar UI
- Controles básicos: Label, Button, Entry, Image, etc.
- Layouts: StackLayout, Grid, FlexLayout son los más usados
- Estilos permiten reutilizar diseño
- Responsive design es crucial para múltiples dispositivos

Conexión con la Siguiente Unidad:

En la Unidad 3 aprenderemos sobre navegación entre páginas y cómo estructurar aplicaciones con múltiples pantallas.

Tarea para Casa

Actividades:

1. **Formulario Completo (50 puntos):**
    - Crear formulario de contacto con Grid
    - Incluir: Nombre, Email, Asunto, Mensaje
    - Usar estilos personalizados
    - Validar campos requeridos
2. **Layout Responsive (30 puntos):**
    - Adaptar formulario para móvil y tablet
    - Usar OnPlatform para diferentes tamaños
3. **Documentación (20 puntos):**
    - Explicar elección de layouts
    - Comentar código XAML

Recursos Adicionales

📚 Material Complementario

Visita nuestra página de recursos para:

- Guías de XAML de Microsoft
- Ejemplos de layouts complejos
- Herramientas de diseño XAML

Ver Recursos →

Recursos Adicionales - Unidad 02

Fundamentos de XAML y Layouts - Enlaces, videos y material complementario

📚 Documentación Oficial de Microsoft

[Guía de XAML en .NET MAUI Oficial]

Documentación completa sobre XAML en .NET MAUI, incluyendo sintaxis, propiedades, y mejores prácticas.

[Layouts en .NET MAUI Oficial]

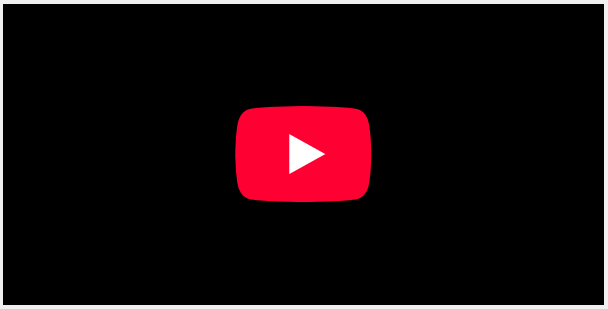
Guía detallada sobre todos los layouts disponibles: StackLayout, Grid, FlexLayout, AbsoluteLayout, etc.

[Controles de .NET MAUI Referencia]

Referencia completa de todos los controles disponibles: Label, Button, Entry, Image, etc.

📹 Videos Educativos

Introducción a XAML



Video tutorial sobre los fundamentos de XAML y cómo crear interfaces de usuario.

[Layouts en .NET MAUI Video]

Tutorial sobre los diferentes layouts disponibles y cuándo usar cada uno.

[Grid Layout Tutorial Tutorial]

Guía detallada sobre cómo usar Grid para crear layouts complejos y responsivos.

🛠️ Herramientas Recomendadas

[XAML Styler Extensión VS]

Extensión para Visual Studio que formatea automáticamente archivos XAML según estándares.

[XAML IntelliSense Extensión]

Mejora el IntelliSense para XAML, ayudando con autocompletado y validación.

📖 Tutoriales y Guías

Grid Layout - Guía Completa

Tutorial detallado sobre Grid, incluyendo RowDefinitions, ColumnDefinitions y posicionamiento.

FlexLayout - Guía Completa

Documentación sobre FlexLayout, el layout moderno basado en CSS Flexbox.

💡 Artículos Especializados

Mejores Prácticas de XAML

Artículo sobre mejores prácticas al escribir XAML para .NET MAUI.

Blog de James Montemagno

Blog con artículos y tutoriales sobre XAML, layouts y desarrollo MAUI en general.

📦 Ejemplos de Código

> [Ejemplos de Layouts GitHub]
>
> Ejemplos de código oficiales mostrando todos los layouts en acción.

> [Ejemplos de Controles GitHub]
>
> Muestras de todos los controles básicos y cómo usarlos.

💡 Consejo de Estudio

Para dominar XAML y Layouts: 1) Practica creando diferentes layouts (Grid, StackLayout, FlexLayout), 2) Experimenta con propiedades y opciones, 3) Usa Hot Reload para ver cambios en tiempo real, 4) Revisa los ejemplos oficiales en GitHub. La práctica constante es clave para entender cómo funcionan los layouts.

Desarrollo de Aplicaciones Multiplataforma con .NET MAUI

Unidad 03: Navegación y Páginas

Notas del Presentador:
La navegación es fundamental en aplicaciones móviles. Esta unidad cubre los diferentes tipos de páginas y sistemas de navegación en MAUI, incluyendo Shell que es el método recomendado.

Competencias y Resultados de Aprendizaje

Resultados de Aprendizaje:

- Conocer los diferentes tipos de páginas en MAUI
- Implementar navegación entre páginas
- Usar Shell Navigation (método recomendado)
- Pasar datos entre páginas
- Implementar deep linking
- Crear menús y barras de navegación

Agenda de la Unidad

1. Tipos de páginas en MAUI
2. ContentPage
3. NavigationPage
4. TabbedPage
5. FlyoutPage (Master-Detail)
6. Shell Navigation (Recomendado)
7. Navegación entre páginas
8. Pasar datos entre páginas
9. Deep linking
10. Práctica guiada

Tipos de Páginas en .NET MAUI

Páginas Disponibles:

- **ContentPage:** Página básica con contenido
- **NavigationPage:** Página con pila de navegación
- **TabbedPage:** Página con pestañas
- **FlyoutPage:** Página con menú lateral (Master-Detail)
- **Shell:** Sistema de navegación moderno (RECOMENDADO)

> **Recomendación:** Shell es el sistema de navegación recomendado por Microsoft para nuevas aplicaciones. Ofrece mejor rendimiento y más funcionalidades.

ContentPage - Página Básica

> **ContentPage:** La página más básica y común. Contiene un solo contenido.

Estructura:

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

    <ContentPage.Content>
        <Label Text="Hola Mundo" />
    </ContentPage.Content>
</ContentPage>
```

Uso:

- Páginas individuales
- Formularios
- Pantallas de contenido
- Base para otras páginas

NavigationPage - Navegación con Pila

> **NavigationPage:** Permite navegación jerárquica con botón de retroceso automático.

Configuración:

```
// En App.xaml.cs o MauiProgram.cs
MainPage = new NavigationPage(new MainPage());

// Navegar a otra página
await Navigation.PushAsync(new SegundaPage());

// Volver
await Navigation.PopAsync();
```

Características:

- Barra de navegación automática
- Botón de retroceso
- Pila de navegación (LIFO)
- Animaciones de transición

TabbedPage - Páginas con Pestañas

> **TabbedPage:** Organiza múltiples páginas en pestañas en la parte inferior (móvil) o superior (escritorio).

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

    <TabbedPage.Children>
        <ContentPage Title="Inicio" IconImageSource="home.png">
            <Label Text="Página de Inicio" />
        </ContentPage>

        <ContentPage Title="Perfil" IconImageSource="user.png">
            <Label Text="Página de Perfil" />
        </ContentPage>

        <ContentPage Title="Configuración" IconImageSource="settings.
            <Label Text="Configuración" />
        </ContentPage>
    </TabbedPage.Children>
</TabbedPage>
```

FlyoutPage - Menú Lateral

> **FlyoutPage:** Página con menú lateral (hamburguesa) que se desliza desde el lado.

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

    <FlyoutPage.Flyout>
        <ContentPage Title="Menú">
            <ListView>
                <ListView.ItemsSource>
                    <x:Array Type="{x:Type x:String}">
                        <x:String>Inicio</x:String>
                        <x:String>Perfil</x:String>
                        <x:String>Configuración</x:String>
                    </x:Array>
                </ListView.ItemsSource>
            </ListView>
        </ContentPage>
    </FlyoutPage.Flyout>

    <FlyoutPage.Detail>
        <NavigationPage>
            <ContentPage Title="Inicio" />
        </NavigationPage>
    </FlyoutPage.Detail>
</FlyoutPage>
```

Shell - Sistema de Navegación Moderno

> **Shell:** Sistema de navegación moderno y recomendado por Microsoft. Combina lo mejor de TabbedPage y FlyoutPage.

Ventajas:

- Mejor rendimiento
- Deep linking integrado
- URI-based navigation
- Menú lateral integrado
- Pestañas inferiores
- Búsqueda integrada

> **Recomendación:** Usa Shell para todas las aplicaciones nuevas. Es más moderno y potente.

AppShell.xaml - Estructura Básica

```
<Shell x:Class="MiApp.AppShell"
       xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
       xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       xmlns:local="clr-namespace:MiApp"
       Shell.FlyoutBehavior="Flyout">

    <Shell.FlyoutContent>
        <VerticalStackLayout Padding="20">
            <Label Text="Mi App" FontSize="24" />
            <Button Text="Inicio" Clicked="OnInicioClicked" />
            <Button Text="Perfil" Clicked="OnPerfilClicked" />
        </VerticalStackLayout>
    </Shell.FlyoutContent>

    <TabBar>
        <ShellContent Title="Inicio"
                      ContentTemplate="{DataTemplate local:MainPage}"
                      Icon="home.png" />
        <ShellContent Title="Perfil"
                      ContentTemplate="{DataTemplate local:ProfilePag
                      Icon="user.png" />
    </TabBar>
</Shell>
```

Navegación con Shell

Navegación por URI:

```
// Navegar a una página específica
await Shell.Current.GoToAsync("//perfil");

// Navegar con parámetros
await Shell.Current.GoToAsync($"//detalle?id={itemId}");

// Navegar relativo
await Shell.Current.GoToAsync("..");

// Navegar con query parameters
await Shell.Current.GoToAsync("//detalle?nombre=Juan&edad=25");
```

Rutas Registradas:

```
// En AppShell.xaml.cs
Routing.RegisterRoute("detalle", typeof(DetallePage));
Routing.RegisterRoute("configuracion", typeof(ConfigPage));
```

Pasar Datos entre Páginas

Método 1: Query Parameters (Shell)

```
// Enviar datos
await Shell.Current.GoToAsync($"//detalle?id={itemId}&nombre={nombre}");

// Recibir datos en la página destino
[QueryProperty(nameof(ItemId), "id")]
[QueryProperty(nameof(Nombre), "nombre")]
public partial class DetallePage : ContentPage
{
    public string ItemId { get; set; }
    public string Nombre { get; set; }

    protected override void OnAppearing()
    {
        base.OnAppearing();
        // Usar ItemId y Nombre
    }
}
```

Pasar Datos - Método 2: Constructor

Pasar objeto completo:

```
// Definir modelo
public class Producto
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public decimal Precio { get; set; }
}

// Navegar pasando objeto
var producto = new Producto { Id = 1, Nombre = "Laptop", Precio = 999
await Navigation.PushAsync(new DetallePage(producto));

// Recibir en constructor
public partial class DetallePage : ContentPage
{
    private Producto _producto;

    public DetallePage(Producto producto)
    {
        InitializeComponent();
        _producto = producto;
        // Usar _producto
    }
}
```

Deep Linking

> **Deep Linking:** Permite abrir la aplicación directamente en una página específica mediante URL.

Configuración (Android):

```
<!-- AndroidManifest.xml -->
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="miapp" />
    </intent-filter>
</activity>
```

Manejar en AppShell:

```
// En AppShell.xaml.cs
protected override async void OnAppearing()
{
    base.OnAppearing();

    // Manejar deep link
    var uri = App.Current.RequestedTheme; // Ejemplo
    if (uri != null)
    {
        await Shell.Current.GoToAsync(uri);
    }
}
```

Menús y Barras de Navegación

Barra de Navegación en Shell:

```
<ShellContent Title="Inicio"
              Route="inicio"
              ContentTemplate="{DataTemplate local:MainPage}"
              Icon="home.png">
    <ShellContent.MenuItems>
        <MenuItem Text="Configuración"
                  IconImageSource="settings.png"
                  Command="{Binding ConfigCommand}" />
        <MenuItem Text="Ayuda"
                  IconImageSource="help.png"
                  Clicked="OnAyudaClicked" />
    </ShellContent.MenuItems>
</ShellContent>
```

Toolbar Items:

```
<ContentPage.ToolbarItems>
    <ToolbarItem Text="Guardar"
                 IconImageSource="save.png"
                 Clicked="OnGuardarClicked" />
    <ToolbarItem Text="Compartir"
                 IconImageSource="share.png"
                 Clicked="OnCompartirClicked" />
</ContentPage.ToolbarItems>
```

Práctica Guiada #3: Aplicación con Navegación

Objetivo:

Crear una aplicación con múltiples páginas y navegación Shell.

Pasos:

1. Crea un proyecto MAUI nuevo
2. Configura AppShell con:
    - Flyout menu con opciones
    - TabBar con 3 pestañas
    - Páginas: Inicio, Perfil, Configuración
3. Implementa navegación entre páginas
4. Pasa datos entre páginas usando QueryProperty
5. Agrega ToolbarItems en una página

Mini-Quiz

Pregunta 1:

¿Cuál es el sistema de navegación recomendado por Microsoft para .NET MAUI?

Respuesta: Shell es el sistema de navegación recomendado. Ofrece mejor rendimiento, deep linking integrado y más funcionalidades.

Pregunta 2:

¿Cómo se pasa un parámetro usando QueryProperty en Shell?

Respuesta: Se navega con query string: GoToAsync("//pagina?param=valor") y se define [QueryProperty(nameof(Propiedad), "param")] en la página destino.

Resumen de la Unidad

Puntos Clave:

- ContentPage es la página básica más común
- NavigationPage permite pila de navegación
- TabbedPage organiza páginas en pestañas
- FlyoutPage ofrece menú lateral
- Shell es el sistema recomendado (más moderno)
- Deep linking permite abrir páginas específicas
- QueryProperty facilita pasar datos entre páginas

Conexión con la Siguiente Unidad:

En la Unidad 4 aprenderemos sobre Data Binding y MVVM, que complementa perfectamente la navegación para crear aplicaciones bien estructuradas.

Tarea para Casa

Actividades:

1. **Aplicación Navegable (60 puntos):**
    - Crear app con Shell (Flyout + TabBar)
    - 3 páginas: Inicio, Lista, Detalle
    - Navegar de Lista a Detalle pasando ID
    - Implementar botón de retroceso
2. **Deep Linking (30 puntos):**
    - Configurar deep linking para Android
    - Probar abrir app con URL personalizada
3. **Documentación (10 puntos):**
    - Explicar estructura de navegación
    - Diagrama de flujo de navegación

Recursos Adicionales

📚 Material Complementario

Visita nuestra página de recursos para:

- Guías de Shell Navigation
- Ejemplos de navegación compleja
- Deep linking avanzado

Ver Recursos →

Recursos Adicionales - Unidad 03

Navegación y Páginas - Enlaces, videos y material complementario

📚 Documentación Oficial de Microsoft

[Shell Navigation en .NET MAUI Oficial]

Guía completa sobre Shell, el sistema de navegación recomendado para .NET MAUI.

[Navegación con Shell Tutorial]

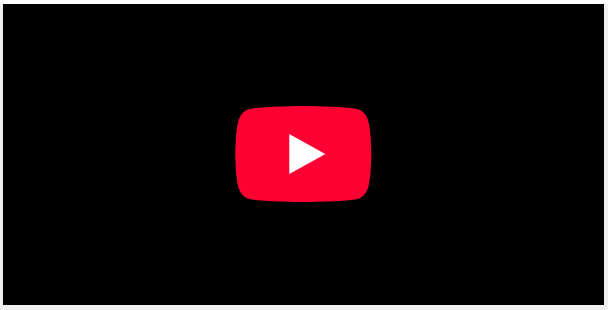
Tutorial detallado sobre cómo implementar navegación usando Shell, incluyendo deep linking.

[Tipos de Páginas Referencia]

Documentación sobre ContentPage, NavigationPage, TabbedPage, FlyoutPage y Shell.

🎥 Videos Educativos

Shell Navigation en .NET MAUI



Tutorial completo sobre Shell Navigation y cómo implementarlo en aplicaciones .NET MAUI.

Deep Linking en .NET MAUI Video

Guía sobre cómo implementar deep linking para abrir páginas específicas mediante URLs.

📦 Ejemplos de Código

Ejemplos de Páginas GitHub

Ejemplos oficiales mostrando diferentes tipos de páginas y navegación.

Ejemplos de Shell GitHub

Ejemplos completos de aplicaciones usando Shell Navigation.

💡 Consejo de Estudio

Para dominar la navegación: 1) Empieza con Shell (es el método recomendado), 2) Practica pasando datos entre páginas, 3) Experimenta con

> deep linking, 4) Crea una app con múltiples páginas y diferentes tipos de navegación. La navegación es fundamental en aplicaciones móviles.

Desarrollo de Aplicaciones Multiplataforma con .NET MAUI

Unidad 04: Data Binding y MVVM

Notas del Presentador:
MVVM es el patrón arquitectónico fundamental en .NET MAUI. Esta unidad es crucial para entender cómo separar la lógica de negocio de la UI.

Competencias y Resultados de Aprendizaje

Resultados de Aprendizaje:

- Comprender el concepto de Data Binding
- Conocer los diferentes modos de binding
- Implementar INotifyPropertyChanged
- Aplicar el patrón MVVM
- Usar Commands (ICommand)
- Crear y usar Value Converters
- Binding con colecciones y ListView/CollectionView

Agenda de la Unidad

1. Concepto de Data Binding
2. Binding Modes (OneWay, TwoWay, OneTime)
3. INotifyPropertyChanged
4. Patrón MVVM (Model-View-ViewModel)
5. Commands (ICommand)
6. Converters (Value Converters)
7. Collections y ListView/CollectionView
8. Binding a servicios y datos
9. Práctica guiada

¿Qué es Data Binding?

> **Data Binding:** Es el proceso que conecta automáticamente los datos de un objeto (ViewModel) con la interfaz de usuario (View).

Ventajas:

- **Separación de responsabilidades:** UI separada de lógica
- **Actualización automática:** UI se actualiza cuando cambian los datos
- **Menos código:** No necesitas actualizar manualmente cada control
- **Testeable:** Lógica de negocio fácil de probar
- **Mantenible:** Código más organizado y fácil de mantener

Sintaxis Básica:

```
<Label Text="{Binding Nombre}" />
<Entry Text="{Binding Email}" />
<Button Command="{Binding GuardarCommand}" />
```

Binding Modes

Tipos de Binding:

- **OneWay:** Datos fluyen del Source al Target (default)
- **TwoWay:** Datos fluyen en ambas direcciones
- **OneTime:** Datos se establecen una vez al inicio
- **OneWayToSource:** Datos fluyen del Target al Source

Ejemplos:

```
<!-- OneWay (default) -->
<Label Text="{Binding Nombre}" />

<!-- TwoWay (para Entry, Editor) -->
<Entry Text="{Binding Email, Mode=TwoWay}" />

<!-- OneTime (valor fijo) -->
<Label Text="{Binding Version, Mode=OneTime}" />
```

> **Nota:** TwoWay es común para Entry y Editor, donde el usuario puede modificar el valor.

INotifyPropertyChanged

> **INotifyPropertyChanged:** Interfaz que notifica cuando una propiedad cambia, permitiendo que la UI se actualice automáticamente.

Implementación Básica:

```
public class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged([CallerMemberName] strin
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(pr
    }
}

public class MiViewModel : ViewModelBase
{
    private string _nombre;

    public string Nombre
    {
        get => _nombre;
        set
        {
            if (_nombre != value)
            {
                _nombre = value;
                OnPropertyChanged();
            }
        }
    }
}
```

Patrón MVVM

Arquitectura MVVM:

Componentes:

- **Model:** Representa datos y lógica de negocio
- **View:** Interfaz de usuario (XAML)
- **ViewModel:** Conecta Model y View, expone datos y comandos

Ventajas:

- Separación clara de responsabilidades
- Reutilización de ViewModels
- Fácil testing de lógica
- UI y lógica independientes

Ejemplo MVVM Completo

Model:

```
public class Persona
{
    public string Nombre { get; set; }
    public string Email { get; set; }
    public int Edad { get; set; }
}
```

ViewModel:

```
public class PersonaViewModel : ViewModelBase
{
    private Persona _persona;

    public string Nombre
    {
        get => _persona.Nombre;
        set
        {
            _persona.Nombre = value;
            OnPropertyChanged();
        }
    }

    public ICommand GuardarCommand { get; }

    public PersonaViewModel()
    {
        _persona = new Persona();
        GuardarCommand = new Command(Guardar);
    }

    private void Guardar() { /* Lógica */ }
}
```

View con Binding:

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

    <ContentPage.BindingContext>
        <local:PersonaViewModel />
    </ContentPage.BindingContext>

    <VerticalStackLayout Padding="20">
        <Entry Text="{Binding Nombre, Mode=TwoWay}"
               Placeholder="Nombre" />
        <Entry Text="{Binding Email, Mode=TwoWay}"
               Placeholder="Email" />
        <Button Text="Guardar"
                Command="{Binding GuardarCommand}" />
    </VerticalStackLayout>
</ContentPage>
```

Commands (ICommand)

> **ICommand:** Permite vincular acciones (como clicks de botones) a métodos en el ViewModel.

Command Simple:

```
public ICommand GuardarCommand { get; }

public PersonaViewModel()
{
    GuardarCommand = new Command(Guardar);
}

private void Guardar()
{
    // Lógica de guardado
}
```

Command con CanExecute:

```
public ICommand GuardarCommand { get; }

public PersonaViewModel()
{
    GuardarCommand = new Command(Guardar, () => PuedeGuardar);
}

private bool PuedeGuardar => !string.IsNullOrEmpty(Nombre);

private void Guardar() { /* ... */ }
```

Value Converters

> **Value Converter:** Transforma valores durante el binding. Útil para formatear datos, convertir tipos, etc.

Implementar Converter:

```
public class BoolToColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object param
    {
        if (value is bool boolValue && boolValue)
            return Colors.Green;
        return Colors.Red;
    }

    public object ConvertBack(object value, Type targetType, object p
    {
        throw new NotImplementedException();
    }
}
```

Usar en XAML:

```
<ContentPage.Resources>
    <local:BoolToColorConverter x:Key="BoolToColor" />
</ContentPage.Resources>

<BoxView Color="{Binding EsActivo, Converter={StaticResource BoolToCo
```

Collections y CollectionView

ObservableCollection:

```
public class PersonasViewModel : ViewModelBase
{
    public ObservableCollection<Persona> Personas { get; set; }

    public PersonasViewModel()
    {
        Personas = new ObservableCollection<Persona>
        {
            new Persona { Nombre = "Juan", Email = "juan@email.com" },
            new Persona { Nombre = "María", Email = "maria@email.com"
        };
    }
}
```

CollectionView en XAML:

```
<CollectionView ItemsSource="{Binding Personas}">
    <CollectionView.ItemTemplate>
        <DataTemplate>
            <VerticalStackLayout Padding="10">
                <Label Text="{Binding Nombre}" FontSize="18" />
                <Label Text="{Binding Email}" TextColor="Gray" />
            </VerticalStackLayout>
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
```

Binding a Servicios

Inyección de Dependencias:

```
// En MauiProgram.cs
builder.Services.AddSingleton<IDataService, DataService>();
builder.Services.AddTransient<PersonaViewModel>();

// En ViewModel
public class PersonaViewModel : ViewModelBase
{
    private readonly IDataService _dataService;

    public PersonaViewModel(IDataService dataService)
    {
        _dataService = dataService;
        CargarDatos();
    }

    private async void CargarDatos()
    {
        var datos = await _dataService.GetPersonasAsync();
        Personas = new ObservableCollection<Persona>(datos);
    }
}
```

Práctica Guiada #4: Aplicación MVVM

Objetivo:

Crear una aplicación usando MVVM con binding completo.

Pasos:

1. Crea proyecto MAUI
2. Implementa:
    - Model: Producto (Id, Nombre, Precio)
    - ViewModel: ProductosViewModel con ObservableCollection
    - View: Lista de productos con CollectionView
3. Agrega Command para agregar producto
4. Implementa binding TwoWay para formulario
5. Crea converter para formatear precio

Mini-Quiz

Pregunta 1:

¿Qué es INotifyPropertyChanged y por qué es importante?

Respuesta: Interfaz que notifica cuando una propiedad cambia, permitiendo que la UI se actualice automáticamente sin código manual.

Pregunta 2:

¿Cuál es la diferencia entre OneWay y TwoWay binding?

Respuesta: OneWay solo actualiza UI cuando cambia el Source. TwoWay permite que cambios en UI actualicen el Source (útil para Entry, Editor).

Resumen de la Unidad

Puntos Clave:

- Data Binding conecta datos con UI automáticamente
- MVVM separa Model, View y ViewModel
- INotifyPropertyChanged notifica cambios de propiedades
- Commands permiten vincular acciones UI con métodos
- Value Converters transforman datos durante binding
- ObservableCollection notifica cambios en colecciones
- CollectionView muestra listas de datos con binding

Conexión con la Siguiente Unidad:

En la Unidad 5 aprenderemos sobre acceso a datos (almacenamiento local, APIs) y funcionalidades nativas, aplicando MVVM para conectarlos.

Tarea para Casa

Actividades:

1. **Aplicación MVVM Completa (60 puntos):**
    - Crear app con Model, ViewModel, View
    - Lista de tareas (ToDo) con ObservableCollection
    - Commands para agregar/eliminar tareas
    - Binding TwoWay para formulario
2. **Value Converter (25 puntos):**
    - Crear converter para formatear fecha
    - Crear converter para color según estado
3. **Documentación (15 puntos):**
    - Diagrama MVVM de la aplicación
    - Explicar flujo de datos

Recursos Adicionales

📚 Material Complementario

Visita nuestra página de recursos para:

- Guías de MVVM de Microsoft
- Ejemplos avanzados de binding
- Mejores prácticas MVVM

Ver Recursos →

Recursos Adicionales - Unidad 04

Data Binding y MVVM - Enlaces, videos y material complementario

📚 Documentación Oficial de Microsoft

Data Binding en .NET MAUI Oficial

Guía completa sobre data binding, incluyendo binding modes, converters, y mejores prácticas.

Binding Modes Tutorial

Explicación detallada de OneWay, TwoWay, OneTime y OneWayToSource.

Value Converters Guía

Cómo crear y usar value converters para transformar datos durante el binding.

🎥 Videos Educativos

MVVM Pattern en .NET MAUI Video

Tutorial sobre cómo implementar el patrón MVVM en aplicaciones .NET MAUI.

Data Binding Tutorial Tutorial

Guía paso a paso sobre data binding y cómo conectarlo con ViewModels.

📦 Ejemplos de Código

Ejemplos de Data Binding GitHub

Ejemplos oficiales mostrando diferentes tipos de binding y converters.

eShopOnContainers - MVVM Completo Ejemplo

Aplicación de ejemplo completa usando MVVM con ViewModels, Commands y binding.

💡 Consejo de Estudio

MVVM es fundamental en .NET MAUI: 1) Practica creando ViewModels con INotifyPropertyChanged, 2) Implementa Commands para acciones, 3) Crea value converters para formatear datos, 4) Usa ObservableCollection para listas. La separación de responsabilidades es clave para aplicaciones mantenibles.

Desarrollo de Aplicaciones Multiplataforma con .NET MAUI

Unidad 05: Acceso a Datos y Funcionalidades Nativas

Notas del Presentador:
Esta unidad final integra todo lo aprendido: acceso a datos locales y remotos, y funcionalidades nativas de cada plataforma. Es crucial para aplicaciones reales.

Competencias y Resultados de Aprendizaje

Resultados de Aprendizaje:

- Implementar almacenamiento local (Preferences, SecureStorage)
- Usar SQLite para base de datos local
- Consumir APIs REST con HttpClient
- Acceder a funcionalidades nativas (cámara, geolocalización)
- Gestionar permisos de plataforma
- Publicar aplicaciones en tiendas

Agenda de la Unidad

1. Almacenamiento local: Preferences
2. SecureStorage para datos sensibles
3. SQLite para bases de datos
4. Acceso a APIs REST
5. HttpClient y servicios
6. Funcionalidades nativas: Cámara
7. Geolocalización
8. Notificaciones
9. Permisos de plataforma
10. Publicación de aplicaciones
11. Práctica guiada

Preferences - Almacenamiento Simple

> **Preferences:** Almacenamiento clave-valor simple para datos no sensibles (configuraciones, preferencias).

Usar Preferences:

```
// Guardar valor
Preferences.Set("nombre", "Juan");
Preferences.Set("edad", 25);
Preferences.Set("esActivo", true);

// Leer valor
string nombre = Preferences.Get("nombre", "Default");
int edad = Preferences.Get("edad", 0);
bool esActivo = Preferences.Get("esActivo", false);

// Eliminar
Preferences.Remove("nombre");

// Limpiar todo
Preferences.Clear();
```

Cuándo Usar:

- Configuraciones de usuario
- Preferencias de la app
- Datos simples (no sensibles)

SecureStorage - Datos Sensibles

SecureStorage: Almacenamiento seguro para datos sensibles como tokens, contraseñas, etc.

Usar SecureStorage:

```
// Guardar de forma segura
try
{
    await SecureStorage.SetAsync("token", "abc123xyz");
    await SecureStorage.SetAsync("usuario_id", "12345");
}
catch (Exception ex)
{
    // Manejar error
}

// Leer de forma segura
try
{
    string token = await SecureStorage.GetAsync("token");
    if (token != null)
    {
        // Usar token
    }
}
catch (Exception ex)
{
    // Manejar error
}

// Eliminar
SecureStorage.Remove("token");
```

Nota: SecureStorage usa el KeyStore de cada plataforma (Android KeyStore, iOS Keychain, etc.)

SQLite - Base de Datos Local

> **SQLite:** Base de datos relacional local, ideal para datos estructurados complejos.

Instalación:

```
// NuGet Package
Microsoft.Data.Sqlite
// o
sqlite-net-pcl
```

Modelo con SQLite-Net:

```
using SQLite;

public class Producto
{
    [PrimaryKey, AutoIncrement]
    public int Id { get; set; }

    public string Nombre { get; set; }
    public decimal Precio { get; set; }
    public DateTime FechaCreacion { get; set; }
}
```

Operaciones SQLite

Conexión y Operaciones:

```
public class DatabaseService
{
    private SQLiteAsyncConnection _database;
    private string _dbPath;

    public DatabaseService()
    {
        _dbPath = Path.Combine(
            FileSystem.AppDataDirectory,
            "productos.db3");
        _database = new SQLiteAsyncConnection(_dbPath);
        _database.CreateTableAsync<Producto>().Wait();
    }

    public async Task<List<Producto>> GetProductosAsync()
    {
        return await _database.Table<Producto>().ToListAsync();
    }

    public async Task SaveProductoAsync(Producto producto)
    {
        if (producto.Id == 0)
            await _database.InsertAsync(producto);
        else
            await _database.UpdateAsync(producto);
    }

    public async Task DeleteProductoAsync(Producto producto)
    {
        await _database.DeleteAsync(producto);
    }
}
```

Acceso a APIs REST

HttpClient: Clase para realizar peticiones HTTP a APIs REST.

Configurar HttpClient:

```
// En MauiProgram.cs
builder.Services.AddHttpClient();

// En servicio
public class ApiService
{
    private readonly HttpClient _httpClient;

    public ApiService(HttpClient httpClient)
    {
        _httpClient = httpClient;
        _httpClient.BaseAddress = new Uri("https://api.ejemplo.com/");
    }

    public async Task<List<Producto>> GetProductosAsync()
    {
        var response = await _httpClient.GetAsync("api/productos");
        response.EnsureSuccessStatusCode();
        var json = await response.Content.ReadAsStringAsync();
        return JsonSerializer.Deserialize<List<Producto>>(json);
    }
}
```

HttpClient - Operaciones CRUD

GET, POST, PUT, DELETE:

```
// GET
var productos = await _httpClient.GetFromJsonAsync<List<Producto>>("a

// POST
var nuevoProducto = new Producto { Nombre = "Laptop", Precio = 999.99
var response = await _httpClient.PostAsJsonAsync("api/productos", nue
var producto = await response.Content.ReadFromJsonAsync<Producto>();

// PUT
producto.Precio = 899.99m;
await _httpClient.PutAsJsonAsync($"api/productos/{producto.Id}", prod

// DELETE
await _httpClient.DeleteAsync($"api/productos/{producto.Id}");
```

Cámara - Funcionalidad Nativa

Usar MediaPicker:

```
// Tomar foto
var foto = await MediaPicker.Default.CapturePhotoAsync();

if (foto != null)
{
    using var stream = await foto.OpenReadAsync();
    var imageSource = ImageSource.FromStream(() => stream);
    // Usar imageSource en Image control
}

// Seleccionar foto de galería
var foto = await MediaPicker.Default.PickPhotoAsync();

if (foto != null)
{
    using var stream = await foto.OpenReadAsync();
    // Procesar imagen
}
```

Permisos Requeridos:

- Android: CAMERA, READ_MEDIA_IMAGES
- iOS: NSCameraUsageDescription, NSPhotoLibraryUsageDescription

Geolocalización

Obtener Ubicación:

```
using Microsoft.Maui.Devices.Sensors;

public async Task<Location> GetLocationAsync()
{
    try
    {
        var request = new GeolocationRequest(
            GeolocationAccuracy.Medium,
            TimeSpan.FromSeconds(10));

        var location = await Geolocation.Default.GetLocationAsync(req

        if (location != null)
        {
            Console.WriteLine($"Lat: {location.Latitude}, Lng: {locat
            return location;
        }
    }
    catch (Exception ex)
    {
        // Manejar error
    }
    return null;
}
```

Permisos:

- Android: ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION
- iOS: NSLocationWhenInUseUsageDescription

Notificaciones

Notificaciones Locales:

```
// En Android/iOS específico
#if ANDROID
using AndroidX.Core.App;

var notification = new NotificationCompat.Builder(context, "canal_id"
    .SetContentTitle("Título")
    .SetContentText("Mensaje")
    .SetSmallIcon(Resource.Drawable.icon)
    .Build();

var notificationManager = NotificationManagerCompat.From(context);
notificationManager.Notify(1, notification);
#endif
```

Usar Community Toolkit:

```
// Instalar: CommunityToolkit.Maui
// En MauiProgram.cs
builder.UseMauiCommunityToolkit();

// En código
var notification = new NotificationRequest
{
    Title = "Título",
    Message = "Mensaje",
    Schedule = new NotificationRequestSchedule
    {
        NotifyTime = DateTime.Now.AddSeconds(5)
    }
};

await notification.Show();
```

Permisos de Plataforma

Solicitar Permisos:

```
using Microsoft.Maui.ApplicationModel;

public async Task<PermissionStatus> RequestCameraPermissionAsync()
{
    var status = await Permissions.CheckStatusAsync<Permissions.Camer

    if (status == PermissionStatus.Granted)
        return status;

    if (status == PermissionStatus.Denied &&
        DeviceInfo.Platform == DevicePlatform.iOS)
    {
        // En iOS, el usuario debe ir a Settings
        return status;
    }

    if (Permissions.ShouldShowRationale<Permissions.Camera>())
    {
        // Mostrar explicación al usuario
    }

    status = await Permissions.RequestAsync<Permissions.Camera>();
    return status;
}
```

Permisos Comunes:

- Camera, Location, Storage, Contacts, etc.

Publicación de Aplicaciones

Android (Google Play):

- Firmar APK/AAB con keystore
- Crear cuenta de desarrollador
- Subir a Google Play Console
- Completar información de la app
- Revisar políticas de Google

iOS (App Store):

- Requiere cuenta de desarrollador ($99/año)
- Firmar con certificados de Apple
- Subir via Xcode o Visual Studio
- App Store Connect

Windows (Microsoft Store):

- Crear paquete MSIX
- Subir a Microsoft Partner Center

Firmar Aplicación Android

Crear Keystore:

```
keytool -genkey -v -keystore mi-release-key.keystore
        -alias mi-alias -keyalg RSA -keysize 2048
        -validity 10000
```

Configurar en proyecto:

```
<!-- Platforms/Android/AndroidManifest.xml -->
<application android:label="MiApp"
             android:icon="@mipmap/appicon">
</application>

<!-- .csproj -->
<PropertyGroup>
    <AndroidKeyStore>True</AndroidKeyStore>
    <AndroidSigningKeyStore>mi-release-key.keystore</AndroidSigningKey
    <AndroidSigningKeyAlias>mi-alias</AndroidSigningKeyAlias>
</PropertyGroup>
```

Práctica Guiada #5: App Completa con Datos

Objetivo:

Crear aplicación completa integrando almacenamiento, API y funcionalidades nativas.

Pasos:

1. Crea app "MiTienda" con:
    - Lista de productos desde API REST
    - SQLite para favoritos locales
    - Preferences para configuración
2. Agrega funcionalidades:
    - Tomar foto del producto
    - Mostrar ubicación de tienda
    - Notificación cuando hay ofertas
3. Gestiona permisos correctamente
4. Prepara para publicación

Mini-Quiz

Pregunta 1:

¿Cuándo usar Preferences vs SecureStorage?

Respuesta: Preferences para datos no sensibles (configuraciones). SecureStorage para datos sensibles (tokens, contraseñas).

Pregunta 2:

¿Qué es necesario antes de acceder a la cámara o ubicación?

Respuesta: Solicitar y obtener permisos usando Permissions.RequestAsync<Permissions.Camera>() o Permissions.Location.

Resumen de la Unidad

Puntos Clave:

- Preferences para datos simples no sensibles
- SecureStorage para datos sensibles
- SQLite para bases de datos locales complejas
- HttpClient para consumir APIs REST
- Funcionalidades nativas: cámara, geolocalización, notificaciones
- Siempre solicitar permisos antes de usar funcionalidades nativas
- Publicación requiere firmar y configurar según plataforma

¡Felicidades!

Has completado el curso de .NET MAUI Básico. Ahora tienes las bases para crear aplicaciones multiplataforma completas.

Tarea Final del Curso

Proyecto Final Integrador:

1. **Aplicación Completa (70 puntos):**
    - App con al menos 3 páginas (Shell navigation)
    - MVVM con ViewModels
    - Almacenamiento local (Preferences o SQLite)
    - Consumo de API REST (o datos mock)
    - Al menos 2 funcionalidades nativas (cámara, ubicación, etc.)
    - Manejo de permisos
2. **Documentación (20 puntos):**
    - README con instrucciones
    - Diagrama de arquitectura
    - Explicación de funcionalidades
3. **Presentación (10 puntos):**
    - Demo de 5 minutos
    - Explicar decisiones técnicas

Recursos Adicionales

📚 Material Complementario

Visita nuestra página de recursos para:

- Guías de publicación en tiendas
- Ejemplos de APIs REST
- Mejores prácticas de seguridad
- Recursos para continuar aprendiendo

Ver Recursos →

Recursos Adicionales - Unidad 05

Acceso a Datos y Funcionalidades Nativas - Enlaces, videos y material complementario

📚 Documentación Oficial de Microsoft

Almacenamiento de Datos Oficial

Documentación sobre Preferences, SecureStorage y SQLite en .NET MAUI.

Integración con Plataforma Oficial

Guía sobre acceso a funcionalidades nativas: cámara, geolocalización, notificaciones, etc.

Publicación de Aplicaciones Guía

Instrucciones para publicar aplicaciones en Google Play, App Store y Microsoft Store.

📹 Videos Educativos

SQLite en .NET MAUI Video

Tutorial sobre cómo usar SQLite para bases de datos locales en aplicaciones MAUI.

[Consumir APIs REST Tutorial]

Guía sobre cómo usar HttpClient para consumir APIs REST desde aplicaciones .NET MAUI.

🛠️ Herramientas Recomendadas

[.NET MAUI Community Toolkit NuGet]

Kit de herramientas de la comunidad con helpers para funcionalidades comunes (cámara, notificaciones, etc.).

[Microsoft.Data.Sqlite NuGet]

Paquete oficial de Microsoft para trabajar con SQLite en .NET.

📖 Tutoriales Avanzados

[Usar la Cámara - Guía Completa]

Tutorial detallado sobre cómo acceder a la cámara y procesar imágenes en .NET MAUI.

[Geolocalización - Guía Completa]

Documentación sobre cómo obtener y usar la ubicación del dispositivo.

📦 Ejemplos de Código

Ejemplos de Funcionalidades Nativas GitHub

Ejemplos oficiales mostrando cómo acceder a funcionalidades específicas de cada plataforma.

💡 Consejo de Estudio

Para esta unidad final: 1) Practica con diferentes tipos de almacenamiento (Preferences, SQLite), 2) Crea un servicio que consuma una API REST real o mock, 3) Experimenta con funcionalidades nativas (cámara, ubicación), 4) Siempre solicita permisos correctamente. Esta unidad integra todo lo aprendido anteriormente.
