UNIVERSIDAD DE COLIMA

Antologías Digitales

Antología de ejercicios y prácticas de programación de móviles en .Net MAUI

Armando Román Gallardo

Colima, Colima, México, a 9 de agosto de 2024

DIRECTORIO

Armando Román Gallardo
Autor

Armando Román Gallardo
Presidente de Academia

Dr. Gerardo Emmanuel Cardenas Villa
Director de la Facultad de Telemática

ÍNDICE

Contenido

PRESENTACIÓN

Propósito

Esta antología ha sido diseñada para apoyar a los estudiantes del 5º semestre de la carrera de Ingeniería en Software en la asignatura "Programación de Móviles". El enfoque principal es introducir a los alumnos a los conceptos fundamentales del desarrollo de aplicaciones móviles utilizando **.NET MAUI** (Multi-platform App UI), una plataforma moderna y versátil que permite desarrollar aplicaciones multiplataforma para Android, iOS y Windows desde un único código base.

Objetivos Generales

1. Proveer una guía teórico-práctica que permita a los alumnos dominar los conceptos esenciales del desarrollo móvil.

2. Desarrollar habilidades prácticas en el diseño e implementación de aplicaciones con .NET MAUI, desde conceptos básicos hasta soluciones complejas como CRUD, manejo de bases de datos, y uso de APIs RESTful.

3. Preparar a los estudiantes para enfrentar retos del ámbito profesional, aplicando estándares modernos en el desarrollo de software móvil.

Contenido General

1. **Introducción al Desarrollo de Aplicaciones Móviles con .NET MAUI**
    - Creación de la primera aplicación Android.
    - Fundamentos del diseño de interfaces con XAML.
    - Configuración del entorno de desarrollo.
2. **Ejercicios Prácticos Básicos**
    - Operaciones matemáticas y uso de controles como Picker, RadioButton y CheckBox.
    - Gestión de colecciones con ListView.
3. **Proyectos Avanzados**
    - Desarrollo de una agenda con SQLite.
    - Implementación de una API RESTful con Node.js.
    - Integración de backend con Node.js en aplicaciones móviles .NET MAUI.

4. **Reto Final**
    - Desarrollo de una aplicación completa que combine autenticación y gestión de usuarios con CRUD, conectada a un backend funcional.

Importancia

La programación móvil es una de las áreas más demandadas en la actualidad. Esta antología no solo proporciona un marco académico sólido, sino que también fomenta el aprendizaje autónomo y práctico para que los estudiantes puedan aplicar los conocimientos adquiridos en proyectos reales.

Agradecimientos

A los docentes y la dirección de la Facultad de Telemática por su apoyo en la creación de esta antología. A los estudiantes, por su dedicación en la búsqueda constante del aprendizaje.

Autor

Armando Román Gallardo
Profesor de la asignatura de Programación de Móviles y presidente de la Academia de la Facultad de Telemática.

CAPÍTULO I: INTRODUCCIÓN AL DESARROLLO DE APLICACIONES MÓVILES/ INTRODUCCIÓN AL DESARROLLO EN .NET MAUI

PRACTICA COMO CREAR NUESTRA PRIMERA APLICACIÓN ANDROID CON .NET MAUI

Introducción

La programación móvil es una habilidad esencial en el desarrollo de software moderno, ya que permite crear aplicaciones innovadoras y accesibles para millones de usuarios en todo el mundo. En este contexto, aprender a trabajar con .NET MAUI (Multi-platform App UI) resulta de gran relevancia, ya que esta plataforma permite desarrollar aplicaciones móviles nativas para Android, iOS y otras plataformas desde un único código base en C# y XAML.

La actividad propuesta tiene como propósito introducir a los estudiantes en los conceptos y herramientas fundamentales de .NET MAUI, comenzando por la creación, configuración y ejecución de su primera aplicación móvil. A través de esta práctica, los estudiantes podrán comprender el flujo básico de desarrollo, desde la configuración del entorno de trabajo hasta la implementación de una interfaz gráfica y la ejecución en dispositivos móviles o emuladores.

Esta actividad no solo refuerza los conocimientos adquiridos en la sección de introducción a la programación móvil, sino que también fomenta el aprendizaje práctico, necesario para enfrentar retos futuros en el diseño y desarrollo de aplicaciones robustas y funcionales. Al dominar estos conceptos iniciales, los estudiantes estarán mejor preparados para avanzar hacia proyectos más complejos en el curso y aplicar estas habilidades en contextos reales del mundo profesional.

Objetivo general
Los estudiantes aprenderán los pasos básicos para crear, configurar y ejecutar una aplicación Android utilizando .NET MAUI, desarrollando habilidades prácticas y consolidando los conceptos fundamentales.

Parte 1: Desarrollo Práctico

1. **Preparativos Iniciales**
   - Antes de comenzar, asegúrate de que cada estudiante tenga instalado:
     - **Visual Studio 2022** (con la carga de trabajo "Desarrollo de aplicaciones móviles con .NET MAUI").
     - **SDK de Android** correspondiente.
     - **Emulador de Android** configurado o un dispositivo móvil físico con depuración USB habilitada.
2. **Creación del Proyecto**
   - Los estudiantes deberán:
     1. Abrir Visual Studio y seleccionar "Crear un nuevo proyecto".
     2. Elegir la plantilla **.NET MAUI App**.
     3. Nombrar el proyecto como "PrimeraAppMAUI".
3. **Configuración y Diseño de la Interfaz**
   - Diseñar la interfaz de usuario utilizando el archivo MainPage.xaml:
     - Agregar controles básicos como Label, Button y Entry.
     - Personalizar el diseño usando propiedades como FontSize, HorizontalOptions, y VerticalOptions.

4. **Establecer la Página de Inicio**
   - Configurar App.xaml.cs para establecer MainPage como la página de inicio:

```
csharp
Copiar código
public partial class App : Application
{
    public App()
    {
        InitializeComponent();
        MainPage = new MainPage();
    }
}
```

5. **Ejecución de la Aplicación**
   - Conectar un dispositivo físico o configurar un emulador.
   - Seleccionar el destino de ejecución en la barra de herramientas.
   - Compilar y ejecutar la aplicación para verificar su funcionamiento.

Parte 2: Evaluación Escrita

Tras completar la actividad práctica, los estudiantes responderán las siguientes preguntas:

1. **Herramientas Necesarias**
   - ¿Qué herramientas y programas necesitas instalar antes de comenzar a desarrollar una aplicación con .NET MAUI?
2. **Configuración del Proyecto**
   - ¿Qué nombre le asignaste a tu proyecto y por qué es importante elegir un nombre descriptivo?
3. **Diseño de la Interfaz de Usuario**
   - ¿En qué archivo se diseña la interfaz de usuario en .NET MAUI y cómo lo configuraste para este proyecto?
4. **Página de Inicio**
   - ¿Cómo configuraste la página de inicio de la aplicación y cuál es su propósito?
5. **Ejecución en Dispositivos Móviles**
   - Describe el proceso que seguiste para ejecutar la aplicación en un dispositivo móvil conectado o en un emulador.

Criterios de Evaluación

Aspecto	Puntos
Instalación y configuración	2
Creación y nombramiento del proyecto	1
Diseño básico de la interfaz	2
Configuración de la página de inicio	2
Ejecución exitosa en dispositivo o emulador	3
Total	**10**

CAPÍTULO I: INTRODUCCIÓN AL DESARROLLO DE APLICACIONES MÓVILES/ INTRODUCCIÓN AL DESARROLLO EN .NET MAUI

EJERCICIOS BASICOS DE PROGRAMACION DE APLICACIONES MOVILES RESUELTOS EN .NET MAUI

Introducción

El propósito de los ejercicios resueltos en la sección de introducción a .NET MAUI es proporcionar una experiencia práctica y conceptual que permita a los estudiantes dominar las bases del desarrollo de aplicaciones móviles. Los ejercicios están diseñados para cumplir con dos objetivos principales:

1. Facilitar la Comprensión de Conceptos

Estos ejercicios prácticos ilustran cómo aplicar principios fundamentales de **.NET MAUI**, como el diseño de interfaces con XAML, el uso de controles interactivos (Picker, ListView, Entry, etc.), y la vinculación de eventos en C#. También permiten a los estudiantes explorar cómo gestionar datos dinámicos, interactuar con elementos de la interfaz y organizar el código de manera modular. Esto ayuda a construir una base sólida para resolver problemas similares en escenarios reales.

2. Desarrollar Habilidades Prácticas

Al trabajar con soluciones completas, los estudiantes aprenden estrategias clave de programación, identifican patrones reutilizables y practican la escritura de código eficiente. También adquieren habilidades en el manejo de errores y validación de datos, además de comprender cómo integrar componentes visuales y funcionales de forma coherente en una aplicación móvil.

Los ejercicios cubren diferentes escenarios para ampliar las habilidades de los estudiantes:

- **"Suma de dos números"**: Enseña cómo capturar datos de entrada, procesarlos y mostrar resultados mediante controles básicos como Entry y Button.
- **"Calculadora con RadioButtons"**: Introduce la selección de opciones y el manejo lógico de entradas múltiples, fortaleciendo la programación condicional.
- **"Calculadora con CheckBox"**: Explora cómo manejar varias opciones al mismo tiempo, mostrando resultados dinámicos según las selecciones del usuario.
- **"Calculadora con Picker"**: Integra un control interactivo para seleccionar operaciones, permitiendo una experiencia más amigable al usuario.
- **"Lista de frutas con ListView"**: Enseña cómo trabajar con colecciones dinámicas, mostrar datos en una interfaz estructurada y manejar acciones comunes como agregar, editar y eliminar elementos.

Valor Académico

Estos ejercicios no solo permiten a los estudiantes resolver problemas específicos, sino que también les proporcionan un marco de referencia para crear proyectos más avanzados.

Refuerzan conceptos clave como la manipulación de colecciones, el diseño de interfaces dinámicas y la integración entre lógica de negocio y visualización de datos. Además, fomentan la confianza y la capacidad para abordar retos complejos en el desarrollo de aplicaciones móviles.

Al final, los estudiantes no solo comprenden mejor los fundamentos de .NET MAUI, sino que también están preparados para implementar soluciones prácticas en proyectos del mundo real, consolidando sus competencias en programación móvil.

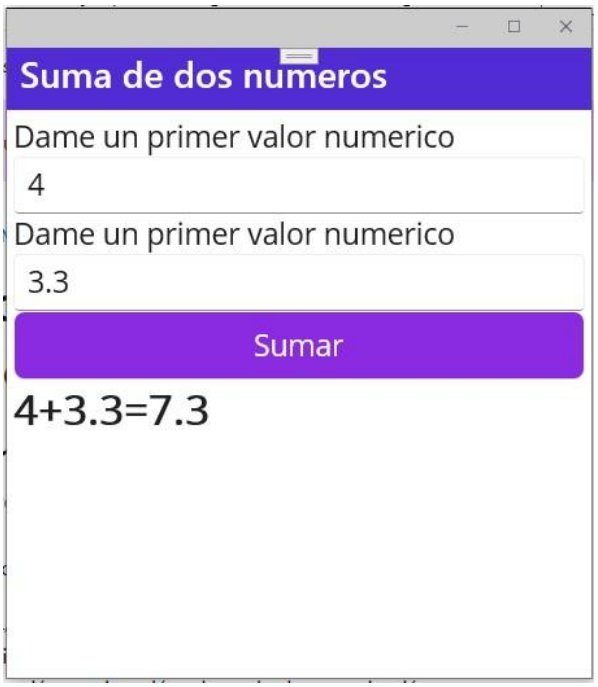
Ejercicio de suma de dos números que utiliza los componentes StackLayout, Label, Entry y Button

Diseño de la interfaz

XAML

```
<?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="SumaDip.MainPage"
Title="Suma de dos numeros">

    <ScrollView>
        <VerticalStackLayout Padding="5">
            <Label Text="Dame un primer valor numerico" FontSize="Medium"/>
            <Entry x:Name="valor1" FontSize="Medium" Keyboard="Numeric" />
            <Label Text="Dame un primer valor numerico" FontSize="Medium"/>
            <Entry x:Name="valor2" FontSize="Medium" Keyboard="Numeric"/>
            <Button Text="Sumar" FontSize="Medium" BackgroundColor="BlueViolet"
                    TextColor="White" Clicked="Button_Clicked"/>
            <Label x:Name="resultado" FontAttributes="Bold" FontSize="Large"/>
        </VerticalStackLayout>
    </ScrollView>

</ContentPage>
```

La aplicación se ve así:



Código fuente de esta aplicación:

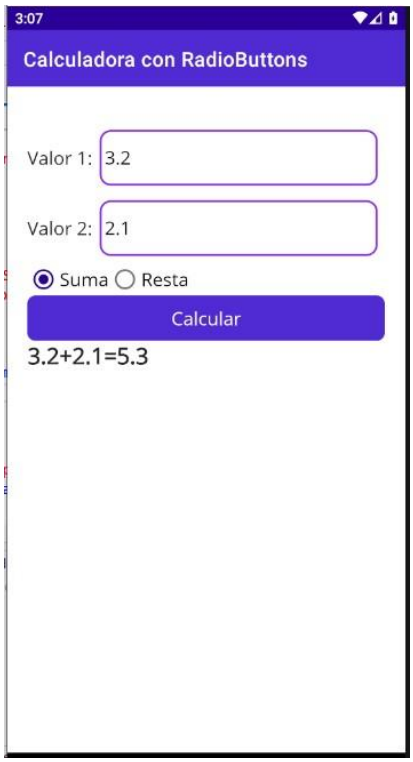
```
namespace SumaDip;

public partial class MainPage : ContentPage
{

       public MainPage()
       {
             InitializeComponent();
       }

    private void Button_Clicked(object sender, EventArgs e)
    {
try         {
            float a = float.Parse(valor1.Text);
float b = float.Parse(valor2.Text);             float c =
a + b;
            resultado.Text = a + "+" + b + "=" + c;
        }
catch         {
            resultado.Text = "Error al capturar los datos!";
        }

    }

}
```

Ejercicio de calculadora de sumas y restas utilizando RadioButtons

Diseño de la interfaz XAML

```
<?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="CalcRadioDip.MainPage"
Title="Calculadora con RadioButtons">

    <ScrollView>
        <StackLayout Margin="20,35,20,25">
            <StackLayout Orientation="Horizontal" VerticalOptions="Center">
                <Label Text="Valor 1:" VerticalOptions="CenterAndExpand"
FontSize="Medium"/>
                <Border Stroke="BlueViolet" StrokeThickness="2"
HorizontalOptions="FillAndExpand" Padding="5" Margin="5">
                    <Border.StrokeShape>
                        <RoundRectangle CornerRadius="10" />
                    </Border.StrokeShape>
                    <Entry x:Name="valor1" Keyboard="Numeric"
HorizontalOptions="FillAndExpand" FontSize="Medium"/>

                </Border>
            </StackLayout>

            <StackLayout Orientation="Horizontal" VerticalOptions="Center" >
                <Label Text="Valor 2:" VerticalOptions="CenterAndExpand"
FontSize="Medium"/>
                <Border Stroke="BlueViolet" StrokeThickness="2"
HorizontalOptions="FillAndExpand" Padding="5" Margin="5">
                    <Border.StrokeShape>
                        <RoundRectangle CornerRadius="10" />
                    </Border.StrokeShape>
                    <Entry x:Name="valor2" Keyboard="Numeric" FontSize="Medium" />
                </Border>
            </StackLayout>

            <StackLayout Orientation="Horizontal">
                <RadioButton x:Name="R1" Content="Suma" IsChecked="True"
GroupName="calculdora" FontSize="Medium"/>
                <RadioButton x:Name="R2" Content="Resta" GroupName="calculdora"
FontSize="Medium"/>
            </StackLayout>

            <Button Text="Calcular"
          Clicked="Button_Clicked" FontSize="Medium"/>
            <Label x:Name="resultado" Text="" FontSize="Large"
FontAttributes="Bold"/>
        </StackLayout>

    </ScrollView>

</ContentPage>
```

Así se ve la interfaz de usuario de la App:



El código fuente de la aplicación:

```
namespace CalcRadioDip;

public partial class MainPage : ContentPage
{


	public MainPage()
	{
		InitializeComponent();
		valor1.Focus();
	}



    private void Button_Clicked(object sender, EventArgs e)     {
            try {
                  float a = float.Parse(valor1.Text);
                  float b =float.Parse(valor2.Text);
                  if (R1.IsChecked)
                  {
                        float c = a + b;
                        resultado.Text = a + "+" + b + "=" + c;
                  }
            if (R2.IsChecked)
            {
                float c = a - b;
```

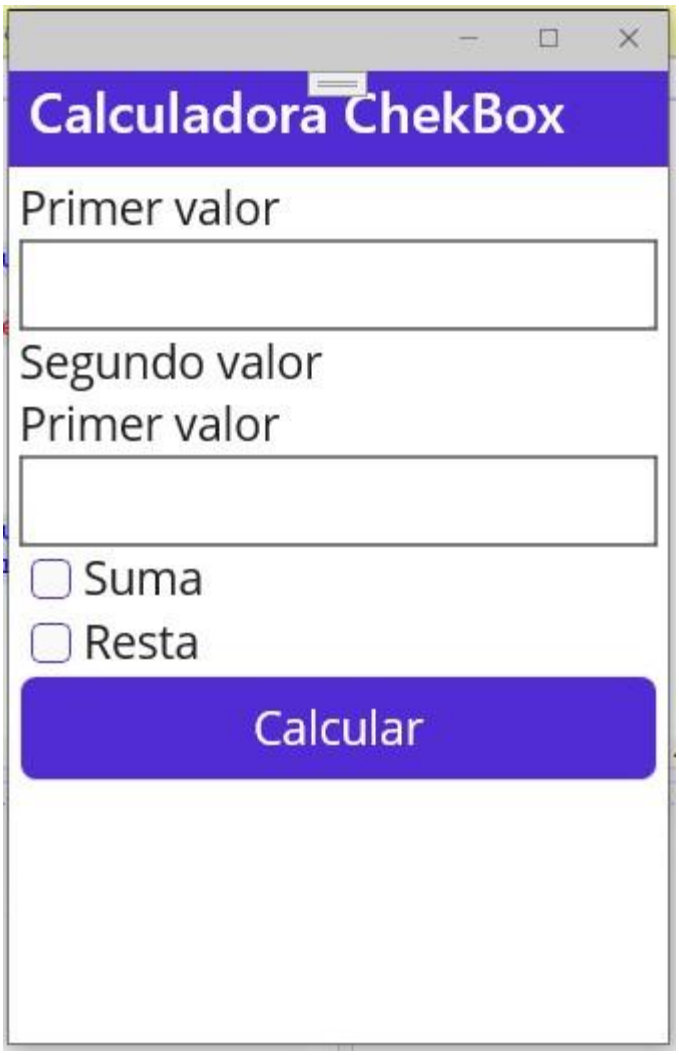
```
                resultado.Text = a + "-" + b + "=" + c;
            }
        }
             catch {
                    resultado.Text = "Error al capturar los datos!!";
             }
    }
}
```

Ejercicio de calculadora de sumas y restas utilizando CheckBox

Diseño de la interfaz XAML

```
<?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:d="http://schemas.microsoft.com/dotnet/2021/maui/design"
xmlns:mc="http://schemas.openxmlformats.org/markupcompatibility/2006"
             x:Class="CalcCheckDip.MainPage"
             Title="Calculadora ChekBox">

    <ScrollView>
        <VerticalStackLayout Padding="5">
            <Label Text="Primer valor" FontSize="Medium"/>
            <Border Stroke="Black" StrokeThickness="1">
            <Entry x:Name="valor1" FontSize="Medium" Keyboard="Numeric"/>
            </Border>

            <Label Text="Segundo valor" FontSize="Medium"/>
            <Label Text="Primer valor" FontSize="Medium"/>
            <Border Stroke="Black" StrokeThickness="1">
            <Entry x:Name="valor2" FontSize="Medium" Keyboard="Numeric"/>
            </Border>

            <HorizontalStackLayout>
                <CheckBox x:Name="suma" />
                <Label Text="Suma" FontSize="Medium">
                    <Label.GestureRecognizers>
                        <TapGestureRecognizer
Tapped="TapGestureRecognizer_Tapped"/>
                    </Label.GestureRecognizers>
                </Label>
            </HorizontalStackLayout>

            <HorizontalStackLayout>
                <CheckBox x:Name="resta" />
                <Label Text="Resta" FontSize="Medium">
                    <Label.GestureRecognizers>
                        <TapGestureRecognizer
Tapped="TapGestureRecognizer_Tapped_1"/>
                    </Label.GestureRecognizers>
                </Label>
            </HorizontalStackLayout>


            <Button Text="Calcular" FontSize="Medium"
Clicked="Button_Clicked"/>
            <Label x:Name="resultado" FontSize="Large" FontAttributes="Bold"/>
        </VerticalStackLayout>
    </ScrollView>

</ContentPage>
```

Así se vería la interfaz de la App



Te muestro el código para interactuar con esta interfaz

```
namespace CalcCheckDip;

public partial class MainPage : ContentPage
{


	public MainPage()
	{
		InitializeComponent();
	}

    private void Button_Clicked(object sender, EventArgs e)
    {	try
{
float a =
float.Parse(valor
1.Text);
float b =
```

```
float.Parse(valor
2.Text);
String s = "";
if
(suma.IsChecked)
            {
                float c = a + b;
                s += a + "+" + b + "=" + c+"\n";
            }
            if (resta.IsChecked)
            {
                float c = a - b;
                s += a + "-" + b + "=" + c;
            }
            resultado.Text = s;
        }
catch {
            resultado.Text = "Error al capturar los datos!!";
        }
    }
    private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
    {
        suma.IsChecked=!suma.IsChecked;
    }
    private void TapGestureRecognizer_Tapped_1(object sender, EventArgs e)
    {
        resta.IsChecked=!resta.IsChecked;
    }
}
```

Ejercicio de calculadora utilizando un Picker

Diseño de la interfaz

XAML

```
<?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="PickerDip.MainPage"
             Title="Calculadora Picker">

    <ScrollView>
        <VerticalStackLayout Padding="5">

            <Label Text="Numero uno" Margin="15,10"/>
            <Border Stroke="black" StrokeThickness="1">
             <Entry x:Name="valor1" Margin="15,0" FontSize="Large"
Keyboard="Numeric"/>
            </Border>

            <Label Text="Numero dos" Margin="15,10"/>
            <Border Stroke="black" StrokeThickness="1">
            <Entry x:Name="valor2" Margin="15,0" FontSize="Large"
Keyboard="Numeric"/>
            </Border>
```

```
            <StackLayout Orientation="Horizontal" Margin="15">
                <Label Text="Selecciona una operación:"
VerticalOptions="Center" />
                <Picker x:Name="operacion" Title="Operaciones"
VerticalOptions="Center">
                    <Picker.Items>
                        <x:String>Suma</x:String>
                        <x:String>Resta</x:String>
                    </Picker.Items>
                </Picker>
            </StackLayout>
            <Button Text="Ejecutar"
            Clicked="Button_Clicked"
                    Margin="10"/>
            <Label x:Name="resultado"
Margin="15,0"
Text="Resultado"
FontAttributes="Bold"
FontSize="Large"/>
        </VerticalStackLayout>
    </ScrollView>

</ContentPage>
```

La aplicación se ve así:

Código fuente de esta aplicación:

```
namespace PickerDip;

public partial class MainPage : ContentPage
{


       public MainPage()
       {
              InitializeComponent();
       }


    private void Button_Clicked(object sender, EventArgs e)
    {
try         {
            float a = float.Parse(valor1.Text);
float b = float.Parse(valor2.Text);             float c;
            switch (operacion.SelectedItem.ToString())
            {                   case
"Suma":                     {
c = a + b;

                        resultado.Text = a + "+" + b + "=" + c;

                        break;
}                   case "Resta":
{                         c = a - b;
                        resultado.Text = a + "-" + b + "=" + c;
break;
                    }
            }
}         catch
{
            resultado.Text = "Error en la captura de datos";
        }
    }
}
```

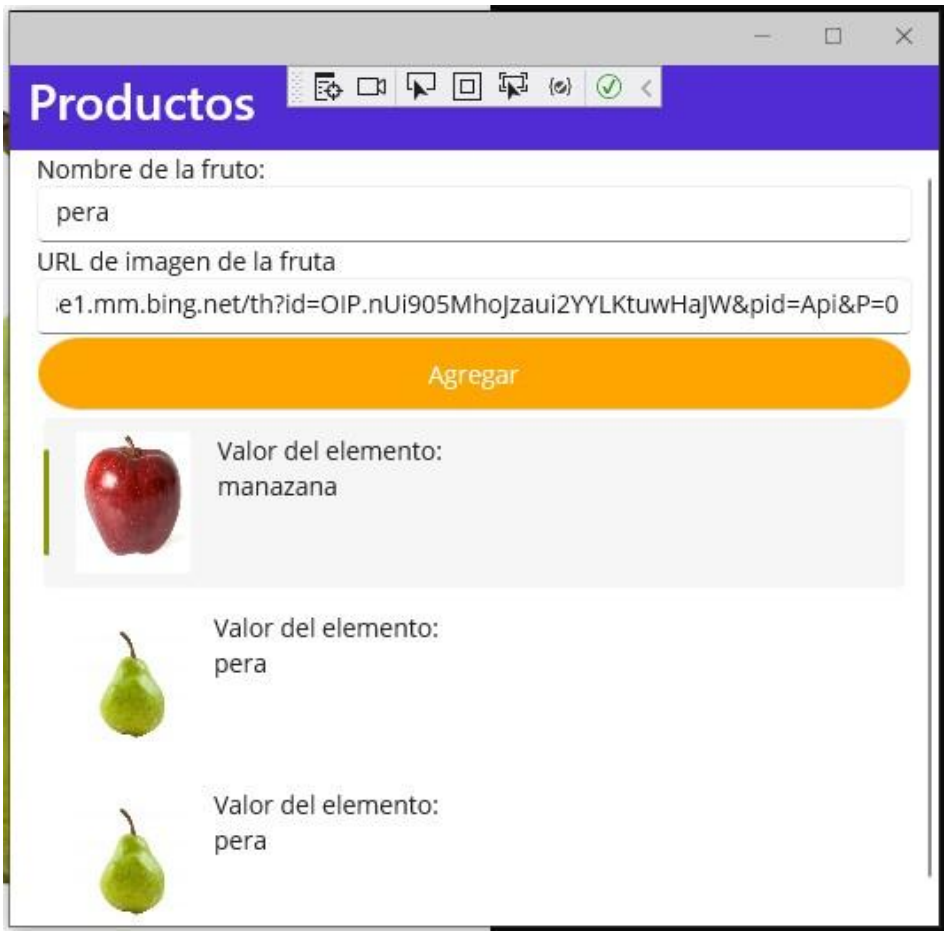
Ejercicio de uso de ListView para mostrar una colección de frutas

Diseño de la interfaz XAML

```
<?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="ListaDip.MainPage"
             Title="Productos">

    <ScrollView>
        <VerticalStackLayout>
            <Label Text="Nombre de la fruto:"
Margin="15,0"/>
            <Entry x:Name="valor" Margin="15,0"/>
            <Label Text="URL de imagen de la fruta" Margin="15,0"/>
```

```
            <Entry x:Name="direccionurl" Margin="15,0"/>
            <Button Text="Agregar" CornerRadius="24" BackgroundColor="Orange"
Clicked="Button_Clicked" Margin="15,1"/>
            <ListView x:Name="milista"
ItemTapped="milista_ItemTapped"
SeparatorColor="Coral" RowHeight="100"
Margin="15,1"
>
                <ListView.ItemTemplate>
                    <DataTemplate >
                        <ViewCell >
                            <ViewCell.ContextActions>
                                <MenuItem Clicked="MenuItem_Clicked" Text="Mostrar"
/>
                                <MenuItem Clicked="MenuItem_Clicked_1"
CommandParameter="{Binding Nombre}"
Text="Borrar" IsDestructive="True" />
                            </ViewCell.ContextActions>
                            <StackLayout Orientation="Horizontal" Padding="10">
                                <Image Source="{Binding Url}"
HeightRequest="80"/>
                                <StackLayout Padding="15,0">
                                    <Label Text="Valor del elemento:"/>
                                    <Label Text="{Binding Nombre}" />
                                </StackLayout>
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>


            </ListView>

        </VerticalStackLayout>
    </ScrollView>

</ContentPage>
>
```

Así se ve la interfaz de usuario de la App:



El código fuente de la aplicación:

```
using System.Collections.ObjectModel;

namespace ListaDip;

public partial class MainPage : ContentPage
{
    public class Fruta
    {
        public string Nombre { get; set; }
public string Url { get; set; }
    }

    ObservableCollection<Fruta> datos = new ObservableCollection<Fruta>();

    public MainPage()
       {
             InitializeComponent();
        milista.ItemsSource = datos;
    }
    private void Button_Clicked(object sender, EventArgs e)
    {
        datos.Add(new Fruta { Nombre = valor.Text, Url = direccionurl.Text });
    }
```

```
    private async void milista_ItemTapped(object sender, ItemTappedEventArgs e)
    {
        var myListView = (ListView)sender;
var myItem = myListView.SelectedItem;         int
index = datos.IndexOf((Fruta)myItem);
        string action = await DisplayActionSheet("Acciones:", "Cancelar",
null, "Eliminar", "Editar");         if (action == "Eliminar")
        {
            datos.RemoveAt(index);
            await DisplayAlert("Eliminar elemento", "Se elimino el lemento no:"
            + index, "OK");
milista.ItemsSource = null;
milista.ItemsSource = datos;
        }
        if (action == "Editar")
        {
            await DisplayAlert("Mensaje", "Seleccio editar", "ok");
        }
    }
    private void MenuItem_Clicked(object sender, EventArgs e)
    {
        DisplayAlert("Mensaje", "Seleccio Mostrar", "ok");
    }
    private void MenuItem_Clicked_1(object sender, EventArgs e)
    {
        var mi = ((MenuItem)sender);
        DisplayAlert("Fruta seleccioada", "Fruta:" +
mi.CommandParameter.ToString(), "ok");
    }
}
```

Al concluir estos ejercicios, es esencial reflexionar sobre los conceptos y habilidades adquiridos, ya que forman los pilares fundamentales para el desarrollo de aplicaciones móviles con **.NET MAUI**. Cada ejercicio no solo ofrece una práctica técnica, sino que también brinda la oportunidad de consolidar aprendizajes clave y preparar el terreno para proyectos más avanzados y desafiantes en el ámbito del desarrollo móvil.

Aspectos Clave para Reflexionar

1. **Diseño de Interfaces de Usuario**
    - ¿Qué descubriste sobre la creación de interfaces con **XAML**, particularmente en ejercicios como la calculadora con Picker y el manejo de listas con ListView?
    - ¿Cómo influye la disposición de los controles en la accesibilidad y la experiencia del usuario?
    - Reflexiona sobre la importancia de diseñar aplicaciones que sean responsivas y funcionales en diferentes tipos de dispositivos móviles.
2. **Captura y Validación de Datos**
    - ¿Cómo garantizaste que los datos introducidos por el usuario fueran correctos, por ejemplo, al seleccionar opciones en un Picker o al interactuar con entradas de texto?
    - ¿Qué estrategias utilizaste para manejar errores en tiempo de ejecución y mejorar la experiencia del usuario?
3. **Interacción Usuario-Programa**

- ¿Qué lógica implementaste para que los controles (como botones, RadioButtons o CheckBox) respondieran a las acciones del usuario?
- ¿Cómo vinculaste eventos de la interfaz de usuario con el código de backend, y cómo mejoraste la interacción entre ambos?

4. **Gestión de Colecciones y Datos Dinámicos**
   - ¿Qué aprendiste sobre la creación y manipulación de colecciones dinámicas, como la lista de frutas en el ejercicio con ListView?
   - Reflexiona sobre cómo la representación y modificación de datos en tiempo real mejora la funcionalidad de las aplicaciones.
5. **Resolución de Problemas Prácticos**
   - ¿Qué técnicas empleaste para depurar el código y resolver problemas durante el desarrollo?
   - ¿Cómo puedes usar estos aprendizajes para abordar desafíos similares en futuros proyectos?
6. **Comparación entre Métodos de Interacción**
   - Analiza las diferencias entre controles como RadioButtons, CheckBox y Picker en términos de experiencia del usuario.
   - ¿Cuáles son las ventajas y desventajas de cada uno según el contexto del ejercicio?

Perspectiva a Futuro

Estos ejercicios representan el primer paso en el camino hacia convertirte en un desarrollador de aplicaciones móviles profesional y competente. Las habilidades adquiridas aquí, como el diseño de interfaces dinámicas, el manejo de datos y la integración de componentes visuales y funcionales, forman la base para proyectos más complejos que pueden incluir bases de datos, API, y diseños de interfaz avanzados.

Cada solución desarrollada no solo resuelve un problema inmediato, sino que también fortalece tu capacidad para enfrentarte a desafíos más grandes y sofisticados en el desarrollo móvil. Recuerda que experimentar y crear más allá de lo aprendido es clave para consolidar tu aprendizaje y alcanzar nuevas metas en el ámbito de la programación con **.NET MAUI**. ¡No temas explorar, innovar y crecer con cada línea de código!

CAPÍTULO I: INTRODUCCIÓN AL DESARROLLO DE APLICACIONES MÓVILES/ INTRODUCCIÓN AL DESARROLLO EN .NET MAUI

PRACTICA DESARROLLO DE UNA APLICACIÓN MÓVIL MULTIPLATAFORMA EN .NET MAUI PARA CAPTURA DE ALUMNOS.

Introducción

En el marco del curso de programación móvil, esta práctica tiene como propósito introducir a los estudiantes en el desarrollo de aplicaciones multiplataforma utilizando .NET MAUI. Esta tecnología permite crear aplicaciones que funcionan tanto en Android como en iOS desde un único código base, facilitando un desarrollo eficiente y moderno.

A través de esta práctica, los estudiantes aplicarán conceptos fundamentales para diseñar, implementar y gestionar interfaces de usuario dinámicas y funcionales, utilizando componentes esenciales como Label, Entry, Button, RadioButton, CheckBox, y Picker. Estas habilidades no solo son fundamentales para proyectos académicos, sino que también preparan a los participantes para retos profesionales reales en el campo del desarrollo móvil.

El ejercicio propone resolver tareas comunes de programación como la captura y gestión de datos. Concretamente, se busca que los estudiantes reflexionen sobre cómo integrar lógica de negocio con diseño de interfaces para crear aplicaciones robustas y centradas en el usuario. Además, se enfatiza la importancia de organizar el código de manera modular y clara, un principio clave para el desarrollo profesional.

Al completar esta práctica, los estudiantes no solo habrán creado una aplicación funcional, sino que también habrán desarrollado habilidades analíticas y técnicas para diseñar soluciones que satisfagan requerimientos específicos. Esto sienta las bases para abordar proyectos más avanzados en el curso y para entender las posibilidades que ofrece .NET MAUI en el desarrollo de aplicaciones móviles.

Objetivo

El objetivo de esta tarea es desarrollar una aplicación móvil en .NET MAUI que permita gestionar un listado de estudiantes, incluyendo funcionalidades para agregar, eliminar y actualizar la información de cada estudiante.

Requisitos:

1.La aplicación debe permitir visualizar un listado de estudiantes con los siguientes datos (utiliza una colección de estudiantes, *no archivos, no base de datos, no API***):**

- Nombre
- Calificación
- Género (Masculino, Femenino, Otro)
- Estado (Activo/Inactivo)
- Municipio

2.Debe haber una sección para agregar un nuevo estudiante, donde el usuario pueda ingresar:

- Nombre
- Calificación (numérica)
- Género (seleccionable mediante un Picker)
- Estado (Activo/Inactivo, mediante un Checkbox)
- Municipio (Colima, Villa de Álvarez, Tecomán, Ixtlahuacán, Coquimatlán, Armería, Comala, Cuauhtémoc, Minatitlán, Manzanillo, mediante un RadioButton)

3. Debe haber una opción para eliminar un estudiante del listado.

4. Debe ser posible actualizar los datos de los estudiantes.

5. Utiliza componentes de interfaz de usuario como Label, Entry, Button, RadioButton, Checkbox y Picker para implementar la aplicación.

6.Maneja la lógica de agregar, actualizar y eliminar estudiantes de una colección en C#.
7.Organiza el código de manera modular y legible, separando la lógica de la interfaz de usuario (C# y XAML).
8.Genera un reporte que incluya: Portada, Introducción, Desarrollo (código y pantallas explicado), Conclusiones (sobre cómo crear el proyecto, los elementos de diseño utilizados, y sobre las colecciones) y súbelo a la plataforma en línea en Formato PDF.
9. Preséntalo en clase al profesor para su evaluación en el horario de la materia cuando el lo solicite.

Rubrica

#	Elementos para evaluar	%
1	Aplicación funcionando que cumpla con todos los requisitos	60 %
2	Reporte en plataforma con los elementos solicitados	20 %
3	Presentarlo al profesor la aplicación en funcionamiento	20 %
	Total	**100%**

Referencias

- Microsoft. (2024). *Introducción a .NET MAUI* | Microsoft Learn. Microsoft. https://learn.microsoft.com/es-es/dotnet/maui/what-is-maui
- Microsoft. (2024). *CollectionView en .NET MAUI* | Microsoft Learn. Microsoft. https://learn.microsoft.com/es-es/dotnet/maui/userinterface/controls/collectionview
- Microsoft. (2024). *Creación de interfaces de usuario con XAML* | Microsoft Learn. Microsoft. https://learn.microsoft.com/es-es/dotnet/maui/user-interface/xaml
- Microsoft. (2024). *Guía de inicio rápido: Creación de una aplicación multiplataforma con .NET MAUI* | Microsoft Learn. Microsoft. https://learn.microsoft.com/eses/dotnet/maui/get-started/first-app

- Instituto Nacional de Estadística y Geografía (INEGI). (2024). *Municipios del estado de Colima*. INEGI. https://www.inegi.org.mx/app/areasgeograficas/?ag=06

CAPÍTULO I: INTRODUCCIÓN AL DESARROLLO DE APLICACIONES MÓVILES/ INTRODUCCIÓN AL DESARROLLO EN .NET MAUI

PRACTICAS CON ARCHIVOS Y BASES DE DATOS EN .NET MAUI

Introducción

Esta práctica tiene como propósito introducir a los estudiantes en el desarrollo de aplicaciones multiplataforma utilizando .NET MAUI, una tecnología moderna y versátil que permite crear aplicaciones para Android, iOS y Windows desde un único proyecto de código compartido.
La práctica se estructura en tres aplicaciones fundamentales, cada una diseñada para abordar conceptos clave de desarrollo móvil, como el manejo de archivos, la autenticación de usuarios, y la interacción con bases de datos locales mediante SQLite. Estas actividades no solo fortalecen las bases técnicas de los estudiantes, sino que también promueven el diseño de interfaces intuitivas y la implementación de funcionalidades esenciales para aplicaciones móviles modernas.
El desarrollo de estas aplicaciones permitirá a los estudiantes explorar y comprender cómo utilizar las APIs de .NET MAUI para gestionar archivos, almacenar preferencias del usuario de manera segura y realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) en bases de datos locales. Además, la práctica culmina con un reto que integra las aplicaciones anteriores, incentivando la investigación y el pensamiento crítico para implementar soluciones más avanzadas.
Con esta experiencia, los estudiantes estarán mejor preparados para enfrentar los desafíos del desarrollo de aplicaciones móviles y adquirirán habilidades prácticas aplicables a escenarios del mundo real.

Objetivo
Desarrollar aplicaciones móviles utilizando .NET MAUI para cumplir con los siguientes requerimientos:

1. Aplicación móvil multiplataforma .NET MAUI para escribir y leer en un archivo de texto: El objetivo es crear una aplicación que permita a los usuarios escribir y guardar texto en un archivo de texto en el dispositivo local. Además, debe permitir leer el contenido de archivos previamente guardados. Esto se puede lograr mediante la implementación de una interfaz de usuario que permita la entrada de texto y botones para guardar y leer el archivo. El acceso al sistema de archivos del dispositivo se realizará utilizando las API proporcionadas por .NET MAUI, y la funcionalidad para escribir y leer en el archivo se implementará utilizando las clases y métodos relevantes.

2. Aplicación móvil multiplataforma .net MAUI para autenticarse utilizando el archivo de preferencia de las aplicaciones para almacenar los datos del usuario en caso de que este así lo decida: El objetivo es crear una aplicación que permita a los usuarios autenticarse en la aplicación utilizando sus credenciales. La aplicación almacenará los datos del usuario, como el nombre de usuario y contraseña, en el archivo de preferencia de la aplicación para su uso posterior. La autenticación se llevará a cabo mediante una interfaz de inicio de sesión que solicitará al usuario sus credenciales y las verificará con los datos almacenados en el archivo de preferencia. En caso de que el usuario lo decida, se guardará la información de inicio de sesión para facilitar futuros accesos.

3. Aplicación móvil multiplataforma .net MAUI para agregar, editar, modificar y buscar información en una agenda utilizando bases de datos locales con SQLite: El objetivo es desarrollar una aplicación que permita a los usuarios gestionar una agenda personal. La aplicación utilizará una base de datos local con SQLite para almacenar los contactos y su información asociada. Los usuarios podrán agregar nuevos contactos, editar la información de los contactos existentes, eliminar contactos y buscar contactos por nombre u otra información relevante. La interfaz de usuario contendrá formularios y controles que permitan realizar estas operaciones de manera intuitiva y sencilla. La aplicación utilizará la biblioteca SQLite-net-pcl para interactuar con la base de datos SQLite y realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

El **objetivo final** es desarrollar estas aplicaciones móviles utilizando .NET MAUI, aprovechando su capacidad de ser multiplataforma y utilizar una sola base de código compartida para desplegar las aplicaciones en Android, iOS y Windows. Además, se utilizará SQLite como base de datos local para almacenar la información necesaria para las funcionalidades de escritura y lectura de archivos de texto, autenticación y gestión de la agenda. Esto permitirá crear aplicaciones móviles completas y conectadas que brinden una experiencia fluida y consistente a los usuarios en diferentes plataformas.

Actividades
Realiza las siguientes aplicaciones .Net Maui Multiplataforma:

Ejercicio de desarrollo de una aplicación móvil multiplataforma .Net MAU para escribir y leer en un archivo de texto

1. Primero, en el archivo MainPage.xaml, actualiza la interfaz de usuario con una Entry para capturar el nombre del archivo, un Editor para capturar el texto y dos botones, uno para guardar el texto y otro para cargarlo:

```
<?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="MauiArchivos.MainPage"
             Title="Editor de texto">

    <ScrollView>
        <StackLayout>
            <Label Text="Nombre del archivo:" />
            <Entry x:Name="fileNameEntry" Placeholder="Ingrese el nombre del archivo" />

            <Label Text="Texto a guardar:" />
            <Editor x:Name="textEditor" HeightRequest="200" />
            <Button Text="Guardar
Texto" Clicked="OnSaveButtonClicked" HorizontalOptions="Center" />
            <Button Text="Cargar
Texto" Clicked="OnLoadButtonClicked" HorizontalOptions="Center" />
            <Label x:Name="resultLabel" HorizontalOptions="Center" VerticalOptions="CenterAndE
xpand" />
        </StackLayout>
    </ScrollView>
```

```
</ContentPage>
```

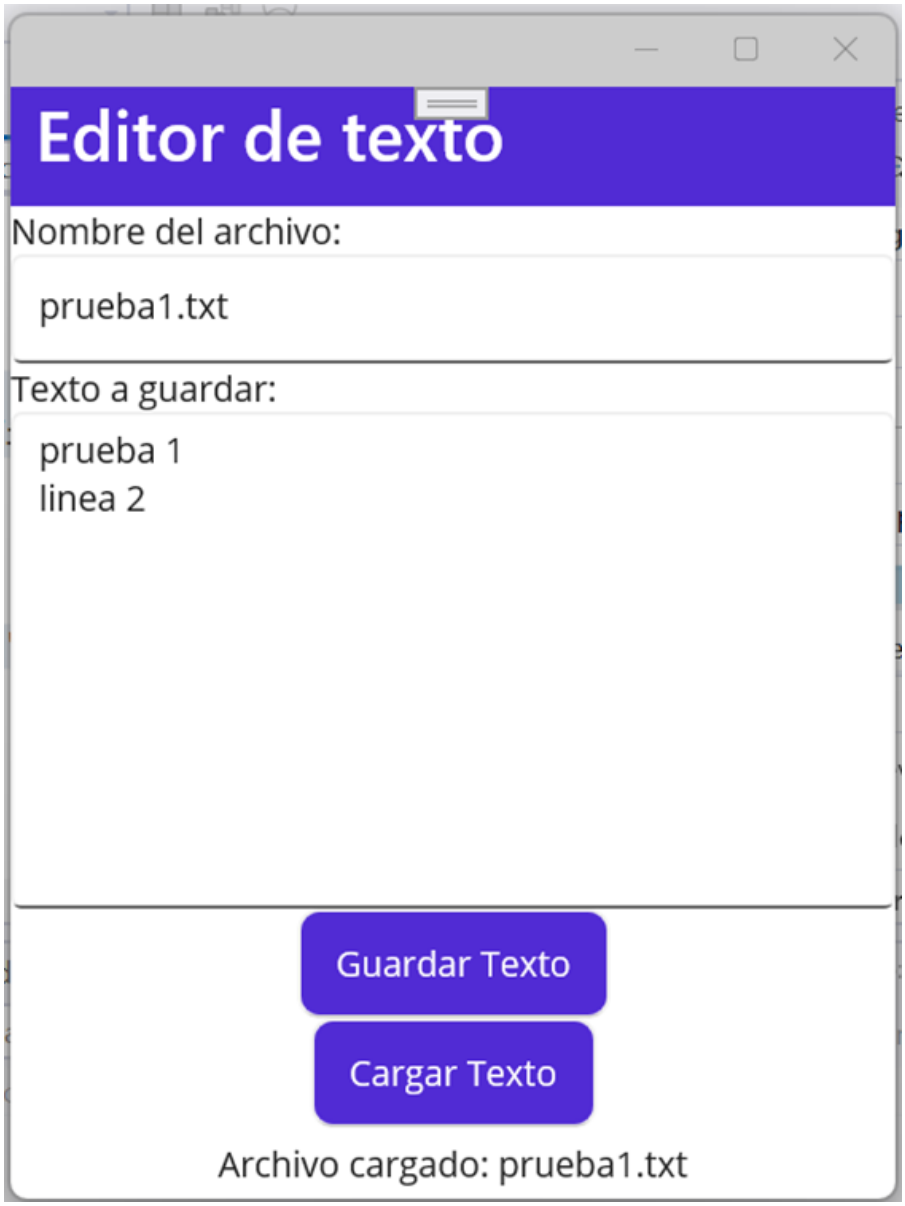
2. Luego, en el archivo MainPage.xaml.cs, implementa la lógica para guardar y cargar el texto:

```
namespace MauiArchivos;
public partial class MainPage : ContentPage
{
    private string filePath;
    public MainPage()
      {
           InitializeComponent();
      }
    private void OnSaveButtonClicked(object sender, EventArgs e)
    {
        // Obtener el nombre del archivo y el texto del editor
        string fileName = fileNameEntry.Text;
        string inputText = textEditor.Text;
        // Validar si el nombre del archivo no está vacío
        if (string.IsNullOrWhiteSpace(fileName))
        {
            DisplayAlert("Error", "Ingrese un nombre de archivo válido.", "Aceptar");
            return;
        }
        // Guardar el texto en el archivo especificado
        filePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
fileName);
        File.WriteAllText(filePath, inputText);
        // Mostrar mensaje de éxito
        DisplayAlert("Texto guardado", "El texto se ha guardado correctamente.", "Aceptar");
    }
    private void OnLoadButtonClicked(object sender, EventArgs e)
    {
        string fileName = fileNameEntry.Text;

        filePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
fileName);
        // Validar si el archivo existe
        if (File.Exists(filePath))
        {
            // Leer el texto desde el archivo
            string savedText = File.ReadAllText(filePath);
            // Mostrar el texto en el editor
            textEditor.Text = savedText;
            // Actualizar etiqueta para mostrar el nombre del archivo cargado
            resultLabel.Text = $"Archivo cargado: {Path.GetFileName(filePath)}";
        }
        else
        {
            // Mostrar mensaje si el archivo no existe
            DisplayAlert("Error", "El archivo especificado no existe.", "Aceptar");
```

```
    }
  }
}
```

Con esta implementación, ahora tienes un formulario donde puedes ingresar el nombre del archivo y el texto a guardar. Al hacer clic en el botón "Guardar Texto", el texto se guardará en el archivo especificado. Luego, al hacer clic en el botón "Cargar Texto", el contenido del archivo se mostrará en el editor y la etiqueta te indicará qué archivo se ha cargado.
Ten en cuenta que, en esta implementación, si deseas cargar el texto de un archivo diferente en el editor, primero debes ingresar el nombre de ese archivo en la Entry antes de hacer clic en el botón "Cargar Texto".

La aplicación se vería de la siguiente manera:



Vista en Windows 11, pero recuerda que la puedes ejecutar en IOS si estas ejecutando el Visual Studio en Mac OSX , o en un dispositivo o emulador de Android en cualquiera de los sistemas operativos.

Ejercicio de desarrollo de una aplicación móvil multiplataforma .net MAUI para autenticarse utilizando que utiliza el archivo de preferencia de las aplicaciones para almacenar los datos del usuario en caso de que este así lo decida.

1. Primero, necesitarás una aplicación .NET MAUI creada con el proyecto de Mobile App (MAUI) en Visual Studio o mediante la CLI. En este ejemplo, utilizaremos Microsoft.Extensions.Configuration para gestionar las preferencias.
2. A continuación, crea una página XAML con el diseño de la pantalla de autenticación (por ejemplo, MainPage.xaml):

```
<?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="MauiLogin.MainPage"
             Title="Login">
    <ScrollView>
        <StackLayout Margin="20">
            <Entry x:Name="UsernameEntry" Placeholder="Username" />
            <Entry x:Name="PasswordEntry" Placeholder="Password" IsPassword="True" />
            <StackLayout Orientation="Horizontal">
                <CheckBox x:Name="RememberMeCheckbox" VerticalOptions="Center" />
                <Label Text="Remember Me" VerticalOptions="Center" />
            </StackLayout>
            <Button Text="Login" Clicked="OnLoginButtonClicked" />
        </StackLayout>
    </ScrollView>
</ContentPage>
```

3. Luego, en el archivo de código detrás de la página (MainPage.xaml.cs), agrega el siguiente código:

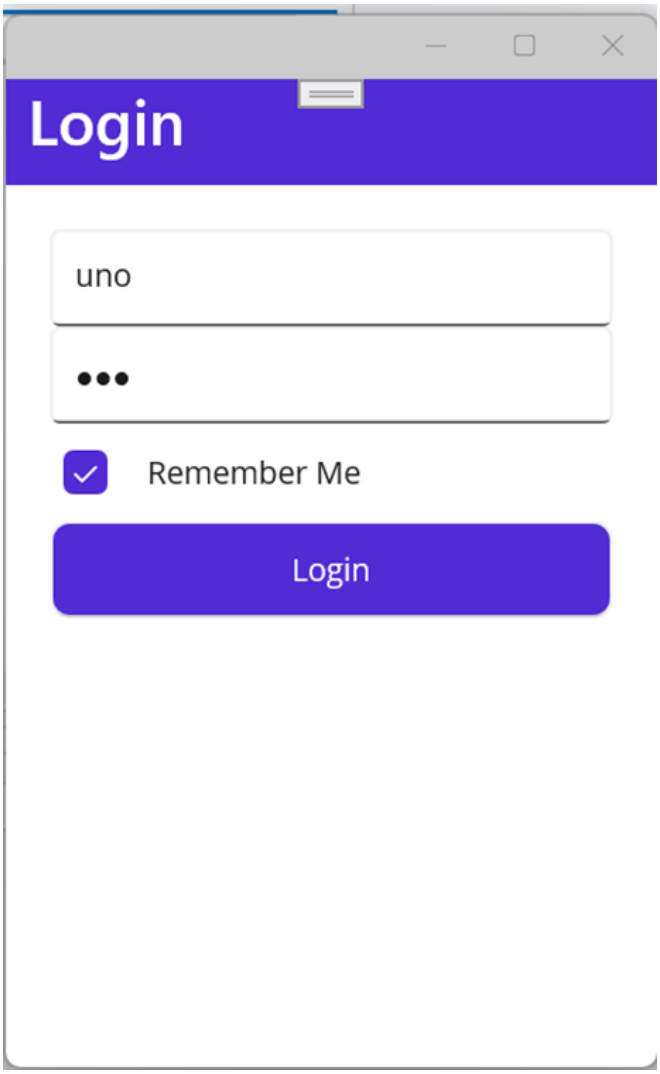
```
namespace MauiLogin;
using Microsoft.Extensions.Configuration;
 public partial class MainPage : ContentPage
{
    private const string RememberMeKey = "RememberMe";
    private const string UsernameKey = "Username";
    private const string PasswordKey = "Password";
    public MainPage()
      {
           InitializeComponent();
        // Cargar datos guardados si están disponibles
        if (Preferences.Get(RememberMeKey, false))
        {
            UsernameEntry.Text = Preferences.Get(UsernameKey, string.Empty);
            PasswordEntry.Text = Preferences.Get(PasswordKey, string.Empty);
            RememberMeCheckbox.IsChecked = true;
        }
    }
    private void OnLoginButtonClicked(object sender, EventArgs e)
    {
```

```
        string username = UsernameEntry.Text;
        string password = PasswordEntry.Text;
        bool rememberMe = RememberMeCheckbox.IsChecked;

        // Aquí realizarías la lógica de autenticación.
        // En este ejemplo, simplemente mostramos un mensaje de éxito.
        if (!string.IsNullOrWhiteSpace(username) && !string.IsNullOrWhiteSpace(password))
        {
            if (rememberMe)
            {
                // Guardar los datos de autenticación en las preferencias
                Preferences.Set(UsernameKey, username);
                Preferences.Set(PasswordKey, password);
                Preferences.Set(RememberMeKey, true);
            }
            else
            {
                // Si no se recuerdan los datos, eliminarlos de las preferencias
                Preferences.Remove(UsernameKey);
                Preferences.Remove(PasswordKey);
                Preferences.Set(RememberMeKey, false);
            }
            // Realizar la lógica de autenticación aquí, por ejemplo, navegación a la siguiente página
            DisplayAlert("Success", "Login successful!", "OK");
        }
        else
        {
            DisplayAlert("Error", "Please enter both username and password.", "OK");
        }
    }
}
```

Con este código, la aplicación permitirá a los usuarios ingresar su nombre de usuario y contraseña, y si seleccionan el checkbox "Remember Me", los datos se guardarán en las preferencias del dispositivo. Cuando la aplicación se inicie la próxima vez, los datos se cargarán automáticamente desde las preferencias si están disponibles.

Recuerda que este ejemplo solo muestra el manejo de la interfaz de usuario y las preferencias. Para una autenticación real, deberás implementar la lógica de autenticación adecuada, como el acceso a una base de datos o una API para verificar las credenciales del usuario.

La aplicación móvil se verá así:



Vista en Windows 11, pero recuerda que la puedes ejecutar en IOS si estas ejecutando el Visual Studio en Mac OSX , o en un dispositivo o emulador de Android en cualquiera de los sistemas operativos.

Ejercicio de desarrollo de una aplicación móvil multiplataforma .net MAUI para agregar, editar, modificar y buscar información en una agenda utilizando bases de datos locales con SQLite.

1. Crear un nuevo proyecto .NET MAUI.Abre Visual Studio o Visual Studio for Mac y crea un nuevo proyecto .NET MAUI con el nombre que desees.
2. Instalar el paquete NuGet SQLite-net-pcl

En el proyecto, haz clic derecho sobre el nodo "Dependencies" y selecciona "Manage NuGet Packages". Busca el paquete "sqlite-net-pcl" e instálalo en el proyecto.

3. Crear el modelo de datos, esta se prondra en el archivo MainPage.axml.cs

```
public class Contacto
   {
      [PrimaryKey, AutoIncrement]
      public int Id { get; set; }
      public string Nombre { get; set; }
      public string Direccion { get; set; }
      public string Telefono { get; set; }
      public string CorreoElectronico { get; set; }
   }
```

4. Crear la base de datos y operaciones CRUD

Crea una clase para gestionar la base de datos SQLite y realizar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar, y Modificar).

```
public class DatabaseManager
   {
      readonly SQLiteConnection database;

      public DatabaseManager(string dbPath)
      {
         database = new SQLiteConnection(dbPath);
         database.CreateTable<Contacto>();
      }
      public List<Contacto> ObtenerContactos()
      {
         return database.Table<Contacto>().ToList();
      }

      public Contacto ObtenerContactoPorNombre(string nombre)
      {
         return database.Table<Contacto>().FirstOrDefault(c => c.Nombre == nombre);
      }

      public int GuardarContacto(Contacto contacto)
      {
         if (contacto.Id != 0)
         {
            return database.Update(contacto);
         }
         else
```

```
        {
            return database.Insert(contacto);
        }
    }
    public int EliminarContacto(Contacto contacto)
    {
        return database.Delete(contacto);
    }
}
```

5. Ahora si pon todo en el archivo MainPage.xaml.cs:

```
namespace MauiBasedeDatosLocal;
using SQLite;
public partial class MainPage : ContentPage
{
    public class Contacto
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }
        public string Nombre { get; set; }
        public string Direccion { get; set; }
        public string Telefono { get; set; }
        public string CorreoElectronico { get; set; }
    }

    public class DatabaseManager
    {
        readonly SQLiteConnection database;

        public DatabaseManager(string dbPath)
        {
            database = new SQLiteConnection(dbPath);
            database.CreateTable<Contacto>();
        }

        public List<Contacto> ObtenerContactos()
        {
            return database.Table<Contacto>().ToList();
        }

        public Contacto ObtenerContactoPorNombre(string nombre)
        {
            return database.Table<Contacto>().FirstOrDefault(c => c.Nombre == nombre);
        }
        public int GuardarContacto(Contacto contacto)
        {
            if (contacto.Id != 0)
            {
                return database.Update(contacto);
            }
            else
```

```
        {
            return database.Insert(contacto);
        }
    }
    public int EliminarContacto(Contacto contacto)
    {
        return database.Delete(contacto);
    }
}
DatabaseManager dbManager;
Contacto contactoEncontrado;
public MainPage()
  {
       InitializeComponent();
    string dbPath = Path.Combine(FileSystem.AppDataDirectory, "contacts.db");
    dbManager = new DatabaseManager(dbPath);
}
private void Guardar_Clicked(object sender, EventArgs e)
{
    if (contactoEncontrado != null)
    {
        // Modificar el contacto encontrado
        contactoEncontrado.Direccion = direccionEntry.Text;
        contactoEncontrado.Telefono = telefonoEntry.Text;
        contactoEncontrado.CorreoElectronico = correoEntry.Text;
        dbManager.GuardarContacto(contactoEncontrado);
        DisplayAlert("Modificar", "Registro Modificado!", "OK");
    }
    else
    {
        // Agregar un nuevo contacto
        var contacto = new Contacto
        {
            Nombre = nombreEntry.Text,
            Direccion = direccionEntry.Text,
            Telefono = telefonoEntry.Text,
            CorreoElectronico = correoEntry.Text
        };
        dbManager.GuardarContacto(contacto);
        DisplayAlert("Agregar", "Registro Agregado!", "OK");
    }
}

private void Buscar_Clicked(object sender, EventArgs e)
{
    string nombre = nombreEntry.Text;
    contactoEncontrado = dbManager.ObtenerContactoPorNombre(nombre);

    if (contactoEncontrado != null)
    {
        direccionEntry.Text = contactoEncontrado.Direccion;
```

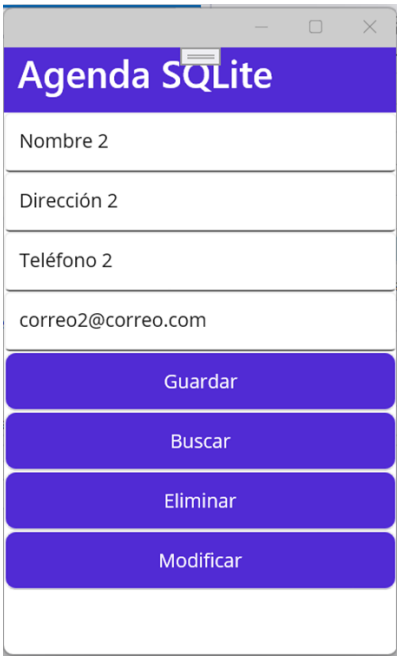
```
            telefonoEntry.Text = contactoEncontrado.Telefono;
            correoEntry.Text = contactoEncontrado.CorreoElectronico;
            ModificarButton.IsEnabled = true;
        }
        else
        {
            // Mostrar un mensaje indicando que el contacto no fue encontrado.
            ModificarButton.IsEnabled = false;
            DisplayAlert("Buscar", "Registro no encontrado!", "OK");
        }
    }
    private void Eliminar_Clicked(object sender, EventArgs e)
    {
        if (contactoEncontrado != null)
        {
            dbManager.EliminarContacto(contactoEncontrado);
            LimpiarCampos();
            DisplayAlert("Eliminar", "Registro eliminado!", "OK");
        }
        else
        {
            // Mostrar un mensaje indicando que el contacto no fue encontrado.
            DisplayAlert("Eliminar", "Registro no encontrado!", "OK");
        }
    }
    private void Modificar_Clicked(object sender, EventArgs e)
    {
        // Habilitar campos para permitir modificaciones
        direccionEntry.IsEnabled = true;
        telefonoEntry.IsEnabled = true;
        correoEntry.IsEnabled = true;
        GuardarButton.IsEnabled = true;
    }
    private void LimpiarCampos()
    {
        nombreEntry.Text = string.Empty;
        direccionEntry.Text = string.Empty;
        telefonoEntry.Text = string.Empty;
        correoEntry.Text = string.Empty;
        contactoEncontrado = null;
        ModificarButton.IsEnabled = false;
        direccionEntry.IsEnabled = false;
        telefonoEntry.IsEnabled = false;
        correoEntry.IsEnabled = false;
        GuardarButton.IsEnabled = false;
    }
}
```

6. Desarrolla la interfaz mediante el archivo MainPage.xaml :

```
<?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="MauiBasedeDatosLocal.MainPage"
            Title="Agenda SQLite">

    <ScrollView>
        <StackLayout>
            <Entry x:Name="nombreEntry" Placeholder="Nombre"/>
            <Entry x:Name="direccionEntry" Placeholder="Dirección"/>
            <Entry x:Name="telefonoEntry" Placeholder="Teléfono"/>
            <Entry x:Name="correoEntry" Placeholder="Correo electrónico"/>
            <Button x:Name="GuardarButton" Text="Guardar" Clicked="Guardar_Clicked"/>
            <Button  Text="Buscar" Clicked="Buscar_Clicked"/>
            <Button Text="Eliminar" Clicked="Eliminar_Clicked"/>
            <Button x:Name="ModificarButton" Text="Modificar" Clicked="Modificar_Clicked" IsEnab
led="False"/>
        </StackLayout>
    </ScrollView>
</ContentPage>
```

La salida de tu aplicación móvil en .net se vera de esta manera:



Vista en Windows 11, pero recuerda que la puedes ejecutar en IOS si estas ejecutando el Visual Studio en Mac OSX , o en un dispositivo o emulador de Android en cualquiera de los sistemas operativos.

Elabora un reporte en formato PDF que demuestre cada uno de estos puntos no olvides estructurarlo:

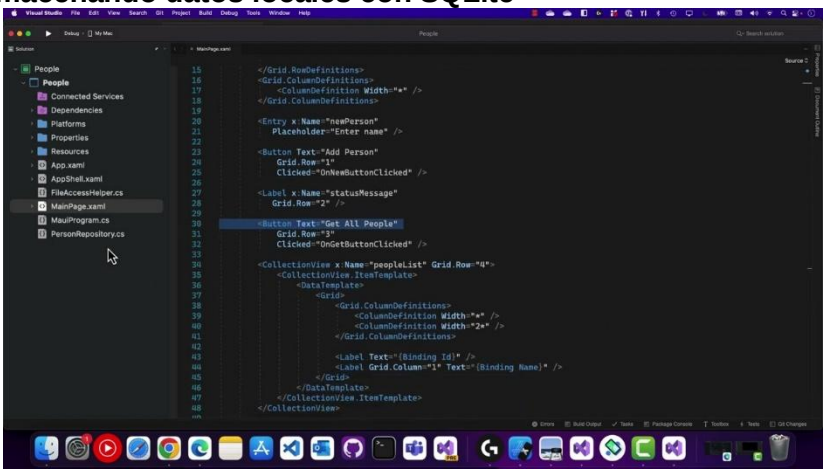
- **Portada**
- ·**Introducción**
- ·**Listado de actividades** realizadas (Evidencia de que desarrollaste las 3 Aplicaciones Solicitadas) con descripción y captura imágenes de apoyo de lo que realizaste (Código Fuente y Pantallas de la Aplicación móvil en Ejecución).

- **Reto:** Desarrolla una aplicación donde juntes la aplicación de autenticación y verifique si es el usuario **uno**, y contraseña **uno** y lo pase a la página de la agenda hecha con sqlite (**Investigar como agregar una página nueva, y navegar hacia esa página**)
- **Conclusiones** (Referente a si se cumplió el objetivo de la actividad)
- **Referencias**

Rubrica

Rubro	Valor
Reporte	60 %
Desarrollo de las 3 aplicaciones	30 %
Desarrollo del Reto	10 %
Total	100%

Videos de Apoyo
.NET MAUI - Almacenando datos locales con SQLite



Reproducir Vídeo en:
https://youtu.be/EitcH1aSivc?si=r_kl6lPDdWKCwXN4

STEP 2 .NET MAUI APP USING SQLITE...HOW TO CREATE .NET MAUI APP CRUD OPERATION USING SQLITE DATABASE
Reproducir video en:
https://youtu.be/f1QOyJclp54
No dudes en consultar las fuentes que se te otorgaron, y preguntar cualquier duda al profesor utilizando los medios de comunicación de esta plataforma. Adelante realiza tu actividad.!!

CAPÍTULO I: INTRODUCCIÓN AL DESARROLLO DE APLICACIONES MÓVILES/ INTRODUCCIÓN AL DESARROLLO EN .NET MAUI

PRACTICA PARA LA IMPLEMENTACIÓN DE UNA API RESTFUL CON NODE.JS, EXPRESS Y SEQUELIZE USANDO MYSQL Y XAMPP

Introducción

Utilizando herramientas como Node.js, Express, Sequelize y MySQL, los estudiantes aprenderán a crear una API funcional que permita realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre un modelo de usuario.
Esta actividad, diseñada para la unidad de introducción a la programación en .NET MAUI, refuerza los conceptos básicos de programación en el backend, estableciendo una base sólida para la integración de servicios web con aplicaciones móviles. Los estudiantes configurarán un entorno de desarrollo local con XAMPP para gestionar la base de datos MySQL y explorarán cómo interactuar con ella mediante Sequelize, un ORM que simplifica las operaciones con bases de datos relacionales.
Adicionalmente, se probará la funcionalidad de la API utilizando herramientas como Postman, fomentando un enfoque práctico y colaborativo. Al finalizar, los estudiantes habrán desarrollado una API robusta que no solo complementa el desarrollo móvil sino que también amplía su comprensión del ecosistema backend y su integración con aplicaciones cliente.

Objetivo:
Implementar una API RESTful que permita realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre un modelo User, utilizando **Node.js**, **Express**, **Sequelize** y una base de datos **MySQL** local a través de **XAMPP**.

Paso 1: Instalación de las Tecnologías

1. **Node.js y npm:**
    - **Instrucciones de instalación:** Instala Node.js desde nodejs.org.
    - **Verifica la instalación:**
    - node -v
    - npm -v
2. **XAMPP:**
    - **Instrucciones de instalación:** Descarga e instala XAMPP desde apachefriends.org.
    - **Configura MySQL:** Inicia XAMPP y activa los servicios **Apache** y **MySQL**. Accede a **phpMyAdmin** a través de http://localhost/phpmyadmin/.
3. **Instalación de Sequelize y MySQL:**
    - **Instrucciones:**
      Navega a la carpeta del proyecto e instala las dependencias necesarias:
    - npm install express sequelize mysql2

Paso 2: Configuración de la Base de Datos MySQL en XAMPP

1. **Acceso a phpMyAdmin:**
   Crea una nueva base de datos llamada pmoviles.

2. **Configuración del modelo de usuario:**
   En la base de datos, se creará una tabla User con las siguientes columnas:

   - id: Entero, llave primaria, autoincremental.
   - nombre: Cadena de texto (nombre del usuario).
   - usuario: Cadena de texto (nombre de usuario único).
   - contrasena: Cadena de texto (contraseña del usuario).

Paso 3: Revisión del Código en servidor.js

servidor.js

```
const express = require('express');
const { Sequelize, DataTypes, Op } = require('sequelize');


const app = express();
app.use(express.json());

// Configurar la conexión a la base de datos
const sequelize = new Sequelize('pmoviles', 'root', '', {
  host: 'localhost',
  dialect: 'mysql'
});

// Definir el modelo de Usuario
const User = sequelize.define('User', {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true
  },
  nombre: {
    type: DataTypes.STRING,
    allowNull: false
  },
  usuario: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true
  },
  contrasena: {
    type: DataTypes.STRING,
```

```
    allowNull: false
  }
});

// Sincronizar el modelo con la base de datos
// { force: false } creará la tabla si no existe, pero no la sobrescribirá si
ya existe
sequelize.sync({ force: false }).then(() => {
  console.log("Base de datos y tablas creadas!");
});

// Ruta única para todas las operaciones CRUD
app.get('/users', async (req, res) => {
  const { action, id, nombre, usuario, contrasena } = req.query;

  try {
    switch(action) {
      case 'create':
        if (!nombre || !usuario || !contrasena) {
          return res.status(400).json({ error: 'Faltan datos requeridos' });
        }
       // const hashedPassword = await bcrypt.hash(contrasena, 10);
        const newUser = await User.create({ nombre, usuario, contrasena});
        return res.status(201).json({ message: 'Usuario creado exitosamente',
id: newUser.id });

      case 'read':
        if (id) {
          const user = await User.findByPk(id, { attributes: ['id', 'nombre',
'usuario'] });
          if (user) {
            return res.json(user);
          } else {
            return res.status(404).json({ message: 'Usuario no encontrado'
});
          }
        } else {
          const users = await User.findAll({ attributes: ['id', 'nombre',
'usuario'] });
          return res.json(users);
        }

      case 'update':
        if (!id) {
          return res.status(400).json({ error: 'Se requiere ID para
actualizar' });
        }
        const user = await User.findByPk(id);
        if (user) {
          if (nombre) user.nombre = nombre;
```

```
          if (usuario) user.usuario = usuario;
          if (contrasena) user.contrasena = contrasena;
          await user.save();
          return res.json({ message: 'Usuario actualizado exitosamente' });
        } else {
          return res.status(404).json({ message: 'Usuario no encontrado' });
        }

      case 'delete':
        if (!id) {
          return res.status(400).json({ error: 'Se requiere ID para eliminar'
});
        }
        const deletedUser = await User.findByPk(id);
        if (deletedUser) {
          await deletedUser.destroy();
          return res.json({ message: 'Usuario eliminado exitosamente' });
        } else {
          return res.status(404).json({ message: 'Usuario no encontrado' });
        }

      default:
        return res.status(400).json({ error: 'Acción no válida' });
    }
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor corriendo en el puerto ${PORT}`);
});
```

El archivo **servidor.js** define un servidor Express y se conecta a la base de datos MySQL utilizando Sequelize.

1. **Conexión a la base de datos:**
   El siguiente fragmento configura Sequelize para conectarse a la base de datos pmoviles que has creado en XAMPP.

```
const sequelize = new Sequelize('pmoviles', 'root', '', {
  host: 'localhost',
  dialect: 'mysql'
});
```

2. **Definición del modelo User:**
   Este código define el modelo User:

```
const User = sequelize.define('User', {
  id: {
    type: DataTypes.INTEGER,
```

```
    autoIncrement: true,
    primaryKey: true
  },
  nombre: {
    type: DataTypes.STRING,
    allowNull: false
  },
  usuario: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true
  },
  contrasena: {
    type: DataTypes.STRING,
    allowNull: false
  }
});
```

3. **Sincronización con la base de datos:**
   Al ejecutar el código, Sequelize sincroniza el modelo con la base de datos, creando la tabla User si no existe:

```
sequelize.sync({ force: false }).then(() => {
  console.log("Base de datos y tablas creadas!");
});
```

Paso 4: Prueba de la API

1. **Inicia el servidor:**
   - Ejecuta el servidor con:
   - node servidor.js

Paso 5: Prueba de Endpoints CRUD utilizando Postman

A continuación, se detalla cómo debes de configurar cada solicitud en **Postman** para probar las operaciones CRUD de la API:

1. Crear un usuario (Create)

- **Método:** GET
- **URL:** http://localhost:3000/users
- **Parámetros (Query Params):**
  - action: create
  - nombre: Nombre del usuario (e.g., Juan Pérez)
  - usuario: Nombre de usuario único (e.g., jperez)
  - contrasena: Contraseña del usuario (e.g., 1234)
- **Ejemplo en Postman:**
  - **GET** http://localhost:3000/users?action=create&nombre=Juan Pérez&usuario=jperez&contrasena=1234

- **Respuesta esperada:**

json

Paso 5: Explicación de las Tecnologías Utilizadas
Al finalizar la actividad, deberás escribir una explicación de:

- **Node.js:** Ambiente de ejecución de JavaScript del lado del servidor.
- **Express:** Framework de Node.js para construir aplicaciones web y APIs.
- **Sequelize:** ORM que facilita las interacciones con bases de datos SQL.
- **MySQL y XAMPP:** Administración de bases de datos relacionales a través de XAMPP y MySQL.
- **Postman.** Probar el endpoint de la API.

Entregables en un solo documento PDF :

1. **Portada.**
2. **API funcional con las rutas CRUD (25%).**
3. **Capturas de pantalla de la base de datos en phpMyAdmin y de las pruebas de los endpoints (25%).**
4. **Documento explicativo de las tecnologías utilizadas (25%).**
5. **Presentarlo al profesor (25%).**

Esta actividad te permitirá trabajar directamente con el archivo servidor.js que ya incluye una configuración sólida para implementar una API RESTful, además de utilizar tecnologías estándar en el desarrollo de backend.

CAPÍTULO I: INTRODUCCIÓN AL DESARROLLO DE APLICACIONES MÓVILES/ INTRODUCCIÓN AL DESARROLLO EN .NET MAUI

EJERCICIO PARA IMPLEMENTAR UNA APLICACIÓN MOVIL DE UN CRUD CON .NET MAUI DONDE EL BACKEND ESTA REALIZADO EN NODE.JS

Introducción

El propósito de este ejercicio es que los estudiantes comprendan e implementen la integración entre una aplicación móvil desarrollada con .NET MAUI y un backend basado en Node.js. A través de la implementación de operaciones CRUD (Crear, Leer, Actualizar y Eliminar), los alumnos aprenderán a interactuar con APIs HTTP y a gestionar datos entre el cliente y el servidor.

En este ejercicio, los estudiantes construirán una interfaz móvil funcional que permita la gestión de usuarios, utilizando controles de entrada, botones y una lista para mostrar los datos en tiempo real. Además, configurarán un servidor Node.js que manejará las operaciones en una base de datos, exponiendo las rutas necesarias para el funcionamiento de la aplicación móvil.

El ejercicio incluye la configuración de Visual Studio para exponer puertos mediante DevTunnels, lo que facilita la conexión entre el frontend y el backend. A lo largo del desarrollo, los alumnos reforzarán conceptos clave como el uso de peticiones HTTP, la depuración de errores en aplicaciones distribuidas, y la implementación de patrones comunes en el desarrollo de software móvil.

Al finalizar este ejercicio, los estudiantes no solo habrán desarrollado una solución completa y funcional, sino que también habrán adquirido experiencia práctica en el diseño e integración de aplicaciones móviles con servicios backend, habilidades fundamentales en el ámbito de la programación móvil.

Objetivo:
El objetivo de este ejercicio es que los alumnos implementen una aplicación CRUD (Crear, Leer, Actualizar, Eliminar) usando .NET MAUI en la parte frontend y un servidor Node.js en la parte backend. Los estudiantes aprenderán a interactuar con APIs HTTP y a usar Visual Studio para exponer el puerto del servidor mediante port forwarding.

Como se vería la aplicación:

Código de la interfaz en XAML MainPage.xaml:

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


        <Grid RowDefinitions=".3*,.7*">
            <Grid BackgroundColor="#4F3F9B" Opacity="0.9"/>
        <Grid ColumnDefinitions="*,*">
            <Image Source="usuarios.png"
                   Aspect="AspectFit"
                   HeightRequest="120"/>
            <Grid Grid.Column="1">
            <Label Text="Catálogo de Usuarios" HorizontalOptions="Center"
                   VerticalOptions="Center"
```

```
            FontSize="Large"
            TextColor="White" />
        </Grid>
    </Grid>

    <Grid Grid.Row="1">
        <RoundRectangle Margin="-5,-30,-5,-2"
                CornerRadius="30,30,0,0"
                Fill="White"/>
        <StackLayout Margin="5,5,5,0">


        <Label Text="Nombre del usuario" />

        <Entry x:Name="nombre" />

        <Label Text="Nombre de usuario" />

        <Entry x:Name="usuario" />

        <Label Text="Contraseña" />

          <Grid HorizontalOptions="StartAndExpand" VerticalOptions="Center" >
            <!-- Entrada de contraseña con botón para mostrar/ocultar -->
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width=".4*" />
              <ColumnDefinition Width=".6*" />
            </Grid.ColumnDefinitions>

            <!-- Entry para la contraseña -->
            <Entry x:Name="contrasena"
          IsPassword="True"
          Grid.Column="0"
          VerticalOptions="Center" />

            <!-- Botón con ícono para mostrar/ocultar la contraseña -->
            <ImageButton Source="eyeicon.png"
             HeightRequest="16"
             Grid.Column="1"
             BackgroundColor="Transparent"
             Clicked="OnTogglePasswordVisibility"
             VerticalOptions="Center" />
          </Grid>
          <HorizontalStackLayout Margin="0,15,0,0">
        <Button Text="Crear" CornerRadius="4" BackgroundColor="Green"
        TextColor="White" Clicked="Button_Clicked" Margin="5"/>
            <Button Text="Eliminar" CornerRadius="4" BackgroundColor="Red"
TextColor="White" Clicked="Button_Delete_Clicked" Margin="5"/>
            <Button Text="Actualizar" CornerRadius="4" BackgroundColor="GreenYellow"
TextColor="Black" Clicked="Button_Edit_Clicked" Margin="5"/>
        </HorizontalStackLayout>
```

```
        <Label Text="" x:Name="resultado" FontSize="Medium" FontAttributes="Bold"/>
        <ListView x:Name="listausuario" ItemsSource="{Binding UsersList}"
           ItemSelected="UsersListView_ItemSelected"  RowHeight="100">
           <ListView.ItemTemplate>
              <DataTemplate>
                 <ViewCell>
                    <StackLayout Orientation="Horizontal">
                       <StackLayout WidthRequest="100" >
                             <Label Text="Id" FontAttributes="Bold"/>
                             <Label Text="{Binding Id}"/>
                             <Label Text="Usuario" FontAttributes="Bold"/>
                             <Label Text="{Binding Usuario}"/>
                          </StackLayout>
                       <StackLayout>
                          <Label Text="Nombre" FontAttributes="Bold"/>
                          <Label Text="{Binding Nombre}" Margin="0,5,0,5"/>
                       </StackLayout>
                    </StackLayout>
                 </ViewCell>
              </DataTemplate>
           </ListView.ItemTemplate>
        </ListView>
     </StackLayout>
  </Grid>
  </Grid>
</ContentPage>
```

Código en C# para interactuar con la interfaz del usuario:

```
using System.Collections.ObjectModel;
using System.Net.Http.Json;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace ProfeUsuarios
{
   public partial class MainPage : ContentPage
   {

      public ObservableCollection<User> UsersList { get; set; }
      private User selectedUser;
      bool isPasswordVisible = false;
      public MainPage()
      {
         InitializeComponent();

         UsersList = new ObservableCollection<User>();
         BindingContext = this;
         CargarUsuarios();
      }
      public async Task LlamadaGetAsync(string url, bool isCreate = false, bool isUpdate = false,
bool isDelete = false)
```

```
    {
        try
        {
            var client = new HttpClient();
            client.BaseAddress = new Uri(url);

            var response = await client.GetAsync(client.BaseAddress);
            response.EnsureSuccessStatusCode();

            if (isCreate)
            {
                var result = await response.Content.ReadFromJsonAsync<CreateUserResponse>();

                if (result != null && result.Id > 0)
                {
                    UsersList.Add(new User
                    {
                        Id = result.Id,
                        Nombre = nombre.Text,
                        Usuario = usuario.Text
                    });

                    await DisplayAlert("Éxito", result.Message, "OK");
                }
            }
            else if (isUpdate)
            {
                var result = await response.Content.ReadFromJsonAsync<CreateUserResponse>();

                if (result != null && result.Id > 0)
                {
                    selectedUser.Nombre = nombre.Text;
                    selectedUser.Usuario = usuario.Text;
                    selectedUser.Contrasena = contrasena.Text;
                    var index = UsersList.IndexOf(selectedUser);
                    if (index != -1)
                    {
                        UsersList[index] = selectedUser;
                    }


                    await DisplayAlert("Éxito", "Usuario actualizado exitosamente", "OK");
                }
            }
            else if (isDelete)
            {
                await DisplayAlert("Éxito", "Usuario eliminado exitosamente", "OK");
                UsersList.Remove(selectedUser);
            }
```

```
            else
            {
                var users = await response.Content.ReadFromJsonAsync<List<User>>();

                UsersList.Clear();
                foreach (var user in users)
                {
                    UsersList.Add(user);
                }
            }

            nombre.Text = "";
            contrasena.Text = "";
            usuario.Text = "";
        }
        catch (Exception ex)
        {
            await DisplayAlert("Error", ex.Message, "OK");
        }
    }

    private void Button_Clicked(object sender, EventArgs e)
    {
        string url = $"https://dbdgbqz8-3000.usw3.devtunnels.ms/users?action=create&nombre={nombre.Text}&contrasena={contrasena.Text}&usuario={usuario.Text}";
        _ = LlamadaGetAsync(url, isCreate: true);
        nombre.Text = "";
        usuario.Text = "";
        contrasena.Text = "";

    }

    private async void CargarUsuarios()
    {
        string url = "https://dbdgbqz8-3000.usw3.devtunnels.ms/users?action=read";
        await LlamadaGetAsync(url, isCreate: false);
    }

    private void UsersListView_ItemSelected(object sender, SelectedItemChangedEventArgs e)
    {
        if (e.SelectedItem != null)
        {
            selectedUser = e.SelectedItem as User;

            nombre.Text = selectedUser.Nombre;
            usuario.Text = selectedUser.Usuario;
            contrasena.Text= selectedUser.Contrasena;
        }
    }
```

```
        private void OnTogglePasswordVisibility(object sender, EventArgs e)
        {
            // Alternar la visibilidad de la contraseña
            isPasswordVisible = !isPasswordVisible;
            contrasena.IsPassword = !isPasswordVisible;

            // Cambiar el ícono según el estado (opcional)
            var imageButton = sender as ImageButton;
            imageButton.Source = isPasswordVisible ? "eyeoffcon.png" : "eyeicon.png";
        }

        private void Button_Edit_Clicked(object sender, EventArgs e)
        {
            if (selectedUser == null)
            {
                DisplayAlert("Error", "Selecciona un usuario para editar", "OK");
                return;
            }

            string url = $"https://dbdgbqz8-
3000.usw3.devtunnels.ms/users?action=update&id={selectedUser.Id}&nombre={nombre.Text}&
usuario={usuario.Text}&contrasena={contrasena.Text}";
            _ = LlamadaGetAsync(url, isUpdate: true);

        }

        private void Button_Delete_Clicked(object sender, EventArgs e)
        {
            if (selectedUser == null)
            {
                DisplayAlert("Error", "Selecciona un usuario para eliminar", "OK");
                return;
            }

            string                    url                    =                    $"https://dbdgbqz8-
3000.usw3.devtunnels.ms/users?action=delete&id={selectedUser.Id}";
            _ = LlamadaGetAsync(url, isDelete: true);

        }

    }

    public class User
    {
        public int Id { get; set; }
        public string Nombre { get; set; }
        public string Usuario { get; set; }
        public string Contrasena { get; set; }
    }

    public class CreateUserResponse
```

```
    {
        public int Id { get; set; }
        public string Message { get; set; }
    }

}
```

Pasos para desarrollarla:

1. **Preparar el entorno**:
    - Instala Visual Studio con soporte para .NET MAUI.
    - Asegúrate de tener Node.js instalado en tu sistema.
    - Descarga el código proporcionado para el servidor Node.js que gestionará los usuarios.
2. **Configurar el servidor Node.js**:
    - Navega a la carpeta del servidor Node.js.
    - Ejecuta **npm install** para instalar las dependencias (no olvides instalar las dependencias necesarias como se vio en la práctica anterior de la API.
    - Inicia el servidor con node app.js o el comando especificado en el código del servidor.
3. **Exponer el puerto del servidor (Visual Studio Forwarding Port)**:
    - En Visual Studio, utiliza la opción **DevTunnels** para exponer el puerto del servidor Node.js (por ejemplo, el puerto 3000).
    - Copia la URL que se genera al exponer el puerto (por ejemplo, https://[nombre].devtunnels.ms) y úsala para las peticiones desde la app .NET MAUI.
4. **Modificar la aplicación .NET MAUI**:
    - Abre el proyecto .NET MAUI en Visual Studio.
    - Asegúrate de que la URL de las peticiones en el código coincide con la URL expuesta mediante el DevTunnel.
    - Revisa el archivo MainPage.xaml.cs, donde se hacen las llamadas HTTP para asegurarte de que la dirección del backend es correcta.
5. **Ejecutar la aplicación**:
    - Inicia la aplicación .NET MAUI.
    - Usa la interfaz para crear, actualizar y eliminar usuarios en la base de datos. Prueba también a listar los usuarios.

- Verifica que las operaciones CRUD se ejecuten correctamente y que el frontend se actualice con los cambios.

6. **Pruebas y correcciones**:
    - Si alguna operación falla, revisa la consola de Visual Studio o de Node.js para identificar errores.
    - Asegúrate de que la conexión con el backend sea estable y esté correctamente configurada.

Consideraciones:

- Asegúrate de que los datos introducidos en los campos de texto son válidos.
- Si ocurre algún error con el servidor o las peticiones HTTP, usa las herramientas de depuración en Visual Studio.
- Familiarízate con la gestión de API en .NET MAUI y cómo se conecta con un backend externo.

Al finalizar este ejercicio, el alumno deberá haber comprendido cómo integrar una aplicación móvil desarrollada con .NET MAUI con un backend basado en Node.js usando llamadas HTTP y haber ejecutado con éxito operaciones CRUD.

CAPÍTULO I: INTRODUCCIÓN AL DESARROLLO DE APLICACIONES MÓVILES/ INTRODUCCIÓN AL DESARROLLO EN .NET MAUI

PRACTICA QUE PERMITE LA CREACIÓN DE GRÁFICOS ESTADISTICOS CON LA LIBRERÍA MICROCHARTS.MAUI

Introducción

En esta práctica, los estudiantes explorarán el uso de la biblioteca **Microcharts.Maui**, una herramienta ligera y versátil para la creación de gráficos en aplicaciones .NET MAUI. El objetivo principal es aprender a integrar y personalizar gráficos de barras y líneas en una aplicación móvil multiplataforma, haciendo uso de datos dinámicos y visuales atractivos que mejoran la experiencia del usuario.
A través de esta actividad, los estudiantes adquirirán habilidades esenciales para el diseño de interfaces enriquecidas que incluyan elementos gráficos, consolidando su conocimiento en tecnologías modernas como SkiaSharp y Microcharts. La práctica también fomenta la creatividad y la experimentación, al incluir un reto adicional que consiste en diseñar un gráfico circular (pie chart) que represente un escenario específico, como ventas de productos o estadísticas personalizadas.
Al finalizar esta práctica, los estudiantes habrán desarrollado una aplicación funcional que no solo presenta datos de forma visualmente impactante, sino que también aprovecha las capacidades de .NET MAUI para desplegar gráficos de alta calidad en múltiples plataformas. Esta actividad es una introducción fundamental a la integración de herramientas gráficas dentro del ecosistema de aplicaciones móviles modernas.

Objetivo
El objetivo de esta práctica es que los estudiantes implementen gráficos visualmente atractivos y dinámicos utilizando la biblioteca **Microcharts.Maui** en aplicaciones .NET MAUI. A través de esta actividad, se busca que los alumnos desarrollen habilidades para crear, configurar y personalizar gráficos de barras, líneas y circulares, integrándolos de manera efectiva en una aplicación móvil multiplataforma. Esto les permitirá interpretar datos de manera visual, mejorar la experiencia de usuario y aplicar conceptos prácticos de diseño y desarrollo de interfaces gráficas en aplicaciones modernas.

Instrucciones:

1. **Crea un nuevo proyecto en .NET MAUI.**
2. **Agrega las dependencias necesarias:**
    - Instala el paquete NuGet SkiaSharp.Views.Maui.Controls versión 2.88.3.
    - Instala el paquete NuGet Microcharts.Maui versión 1.0.0.
3. **Configura el archivo MauiProgram.cs**: Incluye el siguiente código dentro del archivo para habilitar el uso de Microcharts en tu aplicación:

csharp
Copiar código
using Microcharts.Maui;
using Microsoft.Extensions.Logging;

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

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

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

4. **Diseña la Interfaz en MainPage.xaml:** Usa el siguiente código para crear la interfaz de usuario que incluirá dos gráficos:

```
xml
Copiar código
<?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:micro="clr-namespace:Microcharts.Maui;assembly=Microcharts.Maui"
             x:Class="ElGrafiquito.MainPage"
             Title="Gráficos">

   <ScrollView>
      <VerticalStackLayout>
         <micro:ChartView x:Name="chartView" HeightRequest="500" />
         <micro:ChartView x:Name="chartView1" HeightRequest="500" />
      </VerticalStackLayout>
   </ScrollView>
</ContentPage>
```

5. **Código detrás de la página (MainPage.xaml.cs)**: Aquí definirás los datos a graficar y los asignarás a los gráficos de barras y líneas.

```
using Microcharts;
using SkiaSharp;

namespace ElGrafiquito
{
```

```
    public partial class MainPage : ContentPage
    {
        ChartEntry[] entries = new[]
        {
            new ChartEntry(212)
            {
                Label = "Windows",
                ValueLabel = "112",
                Color = SKColor.Parse("#2c3e50")
            },
            new ChartEntry(248)
            {
                Label = "Android",
                ValueLabel = "648",
                Color = SKColor.Parse("#77d065")
            },
            new ChartEntry(128)
            {
                Label = "iOS",
                ValueLabel = "428",
                Color = SKColor.Parse("#b455b6")
            },
            new ChartEntry(514)
            {
                Label = ".NET MAUI",
                ValueLabel = "214",
                Color = SKColor.Parse("#3498db")
            }
        };

        public MainPage()
        {
            InitializeComponent();
            chartView.Chart = new BarChart
            {
                Entries = entries
            };
            chartView1.Chart = new LineChart
            {
                Entries = entries
            };
        }
    }
}
```

6. **Ejecución y entrega:**
    - Ejecuta la aplicación para visualizar los gráficos.
    - Crea un reporte en PDF que contenga una portada, el código fuente, imágenes de la aplicación en ejecución y una breve descripción del proyecto.

Reto:
Después de completar el ejercicio, crea una nueva vista con un **gráfico circular (pie chart)** utilizando datos adicionales que representen categorías personalizadas, como ventas de productos o tareas completadas. Personaliza los colores y etiquetas para que se ajusten a un nuevo escenario.

Rúbrica para Evaluar la Práctica "Creación de Gráficos con Microcharts.Maui"

Criterio	Descripción	Puntos Máximos	Puntos Obtenidos
Configuración del proyecto	El proyecto incluye correctamente las dependencias necesarias (SkiaSharp.Views.Maui.Controls y Microcharts.Maui).	10	
Interfaz de usuario	La interfaz diseñada en MainPage.xaml contiene los gráficos de barras y líneas correctamente configurados.	15	
Implementación del código	El código en MainPage.xaml.cs muestra correctamente los datos en gráficos de barras y líneas, con etiquetas y colores.	20	
Personalización de gráficos	Los gráficos están personalizados con colores y etiquetas que facilitan la interpretación de los datos.	10	
Funcionalidad	La aplicación se ejecuta sin errores y los gráficos se visualizan correctamente en todas las plataformas probadas.	20	
Reto adicional	Se crea una nueva vista con un gráfico circular (pie chart) que incluye datos adicionales, colores y etiquetas personalizados.	15	
Reporte en PDF	El reporte incluye una portada, descripción del proyecto, código fuente, capturas de la aplicación en ejecución y conclusiones.	10	
Creatividad y presentación	La solución muestra un diseño innovador, gráficos atractivos y una buena presentación general del trabajo.	10	
Total		**100**	

Notas adicionales:

- Cada criterio debe evaluarse con base en la claridad, precisión y calidad del trabajo entregado.

- El profesor puede proporcionar retroalimentación específica para cada apartado, ayudando a los estudiantes a mejorar en futuras prácticas.

Conclusiones

La antología presentada proporciona una base sólida en la programación de aplicaciones móviles con .NET MAUI, una plataforma moderna y poderosa que permite el desarrollo multiplataforma desde un único código base. A través de los ejercicios y prácticas incluidas, los estudiantes han podido:

1. **Dominar los conceptos básicos del desarrollo móvil**, incluyendo la creación y configuración de aplicaciones para Android e iOS, el diseño de interfaces responsivas con XAML, y el manejo de eventos en C#.
2. **Implementar proyectos prácticos**, como calculadoras interactivas, agendas con SQLite y aplicaciones que interactúan con APIs RESTful. Estas experiencias han permitido consolidar habilidades técnicas mientras se enfrentan retos reales del desarrollo de software.
3. **Adquirir competencias avanzadas**, como la integración entre frontend y backend, y la gestión de datos dinámicos, preparándolos para escenarios más complejos en el desarrollo profesional.

Lo que sigue para afianzar sus conocimientos
Para continuar afianzando sus habilidades en la programación de aplicaciones móviles, se recomienda:

1. **Explorar características avanzadas de .NET MAUI**:
   - Integración con servicios en la nube como Azure.
   - Uso de notificaciones push.
   - Implementación de almacenamiento seguro y autenticación.
2. **Desarrollar aplicaciones más completas**:
   - Diseñar proyectos que combinen múltiples módulos, como manejo de bases de datos, gráficos estadísticos, y autenticación de usuarios.
   - Trabajar en aplicaciones que utilicen características específicas del dispositivo, como GPS, cámara, y sensores.
3. **Estudiar patrones y arquitecturas de software**:
   - Familiarizarse con patrones como MVVM (Model-View-ViewModel), ampliamente utilizado en .NET MAUI.
   - Aplicar principios de diseño de software para escribir código modular, mantenible y escalable.
4. **Participar en proyectos colaborativos**:
   - Contribuir a proyectos en equipo, lo que permitirá entender cómo integrar componentes de diferentes desarrolladores y usar herramientas de control de versiones como Git.
5. **Mantenerse actualizado**:
   - Seguir las novedades de .NET MAUI y otras tecnologías relacionadas, ya que el desarrollo móvil evoluciona constantemente.
   - Realizar cursos avanzados, talleres y certificaciones que refuercen su conocimiento.

Conclusión Final

El aprendizaje no se detiene aquí. Los fundamentos adquiridos en esta etapa son un trampolín hacia la creación de soluciones más innovadoras y funcionales. Continuar practicando, investigando y desafiándose a resolver problemas más complejos fortalecerá su desarrollo como programadores móviles y abrirá nuevas oportunidades en un mundo laboral altamente competitivo. ¡El futuro en la programación móvil está en sus manos! 🚀

PERTINENCIA
QUE TRANSFORMA