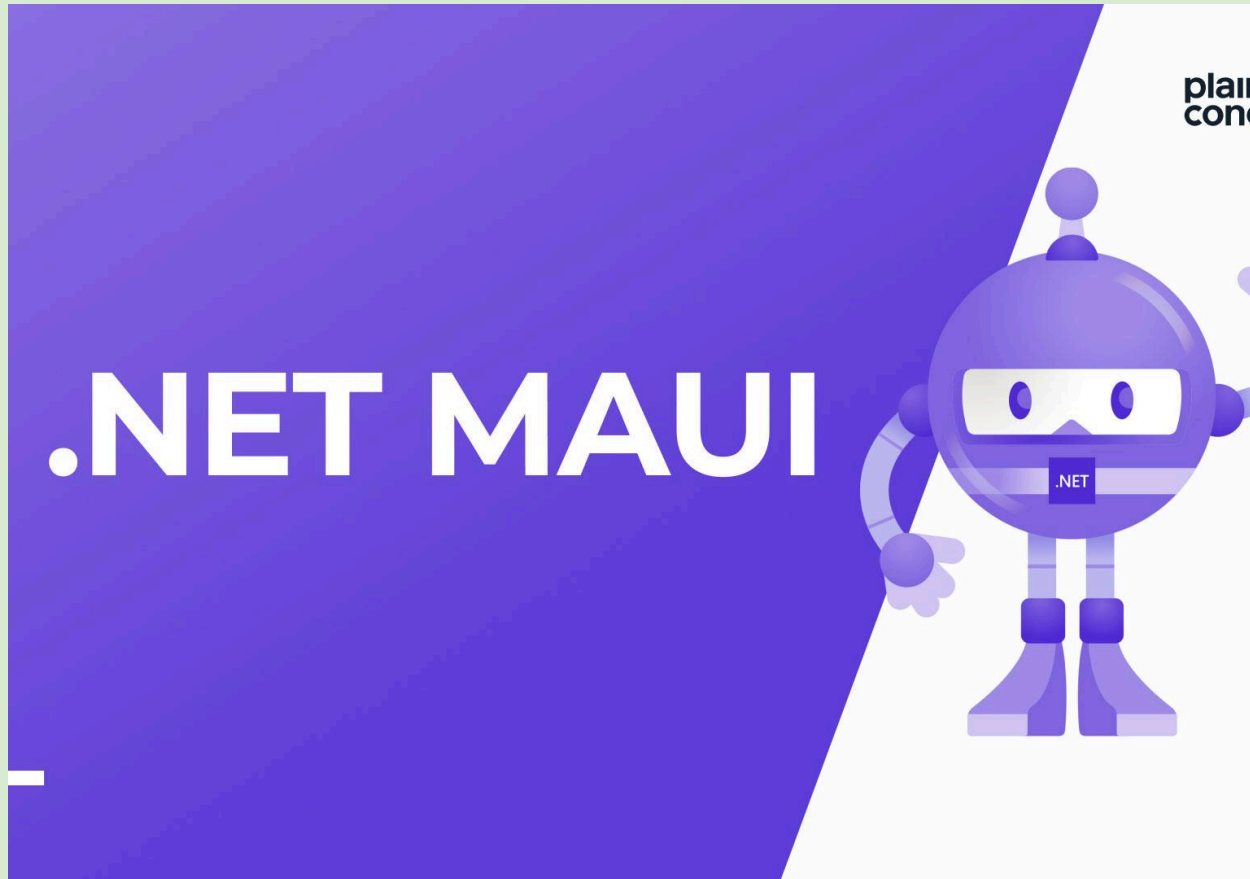


# UNIDAD 04 ACTIVIDAD 02



**Azael Morell Martínez**


09/12/2024

Desarrollo de Interfaces

# ÍNDICE:

---

[ÍNDICE:](#)

[INTRODUCCIÓN](#) 

[OBJETIVOS](#) 

[PLANTEAMIENTOS PREVIOS](#) 

[DIAGRAMA DE GANTT:](#) 

[RESULTADOS](#) 

[VISTA GENERAL DEL PROYECTO:](#)

[SUGERENCIA:](#)

[EL CÓDIGO .XAML:](#)

[EL CÓDIGO .XAML.CS:](#)

[PROBLEMA ENCONTRADO](#) 

[SUGERENCIAS PARA HACER LA APP](#) 

[CONCLUSIÓN](#) 

[WEBGRAFÍA:](#) 

## INTRODUCCIÓN

En este proyecto, desarrollé una aplicación que permite gestionar tareas utilizando Firebase Realtime Database como backend. La idea principal fue crear un gestor sencillo de tareas que reflejara la información en tiempo real y utilizara el patrón MVVM para aprovechar las capacidades de data binding que ofrece .NET MAUI. Además, agregué un toque visual temático navideño para hacerlo más atractivo. Durante el desarrollo, enfrenté algunos desafíos relacionados con la integración de Firebase y la correcta configuración de las dependencias necesarias.

## OBJETIVOS

- **Crear una aplicación funcional de gestión de tareas** que permita a los usuarios agregar, actualizar y eliminar tareas de manera intuitiva.
- **Implementar Firebase Realtime Database** como backend para garantizar la sincronización en tiempo real.
- **Fomentar el aprendizaje de tecnologías** modernas como .NET MAUI y Firebase, enfocándome en su integración.
- **Diseñar una interfaz de usuario atractiva y dinámica** que combine funcionalidad y estética.
- **Resolver los problemas** surgidos durante el desarrollo para mejorar mi comprensión del entorno y la arquitectura.

## PLANTEAMIENTOS PREVIOS

Antes de comenzar, era fundamental tener una comprensión sólida de:

- La estructura y funcionamiento de .NET MAUI.
- Cómo trabajar con XAML para diseñar la interfaz de usuario.
- La implementación de métodos asincrónicos en C#.

## DIAGRAMA DE GANTT:

<u>Tarea</u>	<u>1-3 días</u>	<u>4-7 días</u>	<u>más de una semana</u>
Desarrollo del código y de la interfaz	xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx		
Desarrollo de la documentación	xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx		
Testing del funcionamiento	xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx		

## RESULTADOS 💪

### VISTA GENERAL DEL PROYECTO:

✨ Estoy súper emocionado de compartir el diseño de mi app de empleados. La he llevado a cabo en .NET MAUI y ha quedado genial. ¡Déjame contarte más al respecto!

#### 📝 Diseño de la App:

**Interfaz Colorida** 🎨: El fondo tiene un degradado de naranja a amarillo 🟠🟡, que te pone en el espíritu navideño 🎅.

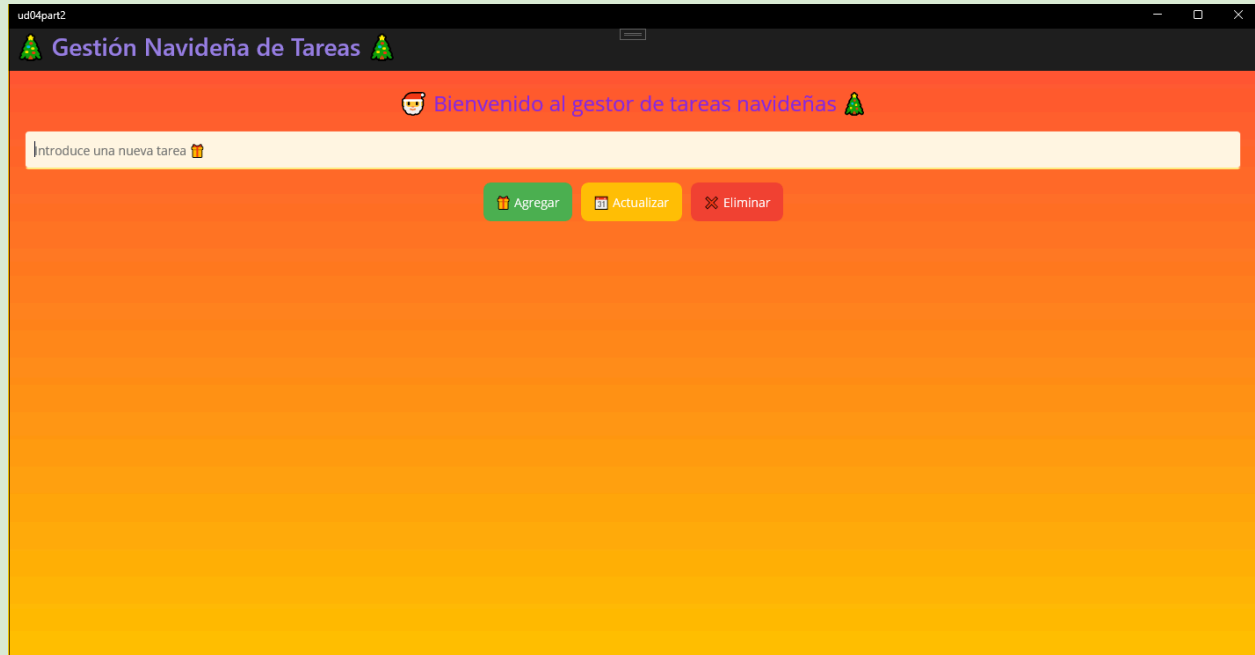
**Título** 🌲: Arriba, el título dice "Gestión Navideña de Tareas" en letras moradas ✨. ¡Muy festivo!

**Bienvenida** 🎁: Justo debajo, un mensaje de bienvenida que dice "Bienvenido al gestor de tareas navideñas" 🎅. ¡Te hace sentir como en casa!

**Barra de Entrada** 🎯: Hay una barra donde puedes "Introducir una nueva tarea" 📝. Súper útil para mantenerte organizado.

#### **Botones Coloridos** 🌟:

- Botón verde que dice "Agregar" ➕.
- Botón amarillo que dice "Actualizar" ↻.
- Botón rojo que dice "Eliminar" ✖.



## SUGERENCIA:

SI QUIERES VER EL FUNCIONAMIENTO DE ESTA APP. RECOMIENDO VER EL VIDEO LLAMADO “TestingAPP\_EMPLEADOS” QUE ESTÁ SUBIDO EN ESTE MISMO REPOSITORIO DE GITHUB.

## EL CÓDIGO .XAML:

---

He desarrollado una aplicación de gestión de tareas navideñas utilizando .NET MAUI. La interfaz de usuario está diseñada para ser intuitiva y festiva, con un fondo de degradado que varía del naranja al amarillo. El encabezado de la aplicación, titulado "Gestión de Tareas", se presenta en un texto de color morado.

**En la parte superior de la interfaz**, se incluye un campo de entrada, definido por la etiqueta <Entry>, destinado a la introducción de nuevas tareas. Este elemento permite al usuario agregar tareas de manera eficiente. Justo debajo del campo de entrada, se dispone de tres botones, definidos por la etiqueta <Button>, con funcionalidades específicas:

- **Agregar tarea:** Botón de color verde, destinado a añadir nuevas tareas a la lista.
- **Actualizar tarea:** Botón de color amarillo, utilizado para modificar tareas existentes.
- **Eliminar tarea:** Botón de color rojo, encargado de eliminar tareas seleccionadas de la lista.

**La lista de tareas** se presenta en un formato estructurado, utilizando la etiqueta <CollectionView>, permitiendo una visualización clara y ordenada de las tareas registradas. Cada tarea se muestra con su correspondiente nombre, facilitando la gestión y organización de las mismas durante la temporada navideña.

```

<?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="ud04part2.MainPage"
  Title="Gestión de Tareas">

  <ContentPage.Content>

    <StackLayout Padding="20" Spacing="15">

      <!-- Entry para ingresar la tarea -->
      <Entry x:Name="TaskEntry" Placeholder="Introduce una nueva tarea" />

      <!-- Botones para agregar, actualizar y eliminar tareas -->
      <StackLayout Orientation="Horizontal" Spacing="10">

        <Button Text="Agregar" Clicked="AddTask" />
        <Button Text="Actualizar" Clicked="UpdateTask" />
        <Button Text="Eliminar" Clicked="DeleteTask" />

      </StackLayout>

      <!-- ListView para mostrar la lista de tareas -->
      <CollectionView x:Name="TaskList"
        ItemsSource="{Binding Tasks}"
        SelectionChanged="OnTaskSelected">

        <CollectionView.ItemTemplate>

          <DataTemplate>

            <StackLayout Padding="10">

              <Label Text="{Binding NombreTarea}" FontSize="18" />

            </StackLayout>

          </DataTemplate>

        </CollectionView.ItemTemplate>

      </CollectionView>

    </StackLayout>
  </ContentPage.Content>

</ContentPage>

```



## EL CÓDIGO .XAML.CS:

### Descripción Técnica:

#### 1. Bibliotecas Importadas:

- Firebase.Database y Firebase.Database.Query: Para interactuar y realizar consultas en Firebase Realtime Database.
- System.Collections.ObjectModel: Para manejar colecciones observables que reflejan cambios automáticamente en la UI.
- System.Linq: Para métodos útiles en la manipulación de colecciones.
- System.Windows.Input: Para utilizar ICommand en la aplicación.
- System: Para manejar eventos y tipos básicos como EventArgs.

#### 2. Configuración Inicial:

- Se define una URL constante que apunta a la base de datos Firebase.
- Se crea un cliente de Firebase para interactuar con la base de datos.
- Se define una colección observable de tareas (ObservableCollection<TaskItem>) que se actualiza automáticamente en la UI.

#### 3. Constructor de la Clase MainPage:

- Se inicializan los componentes definidos en el archivo XAML.
- Se inicializa el cliente de Firebase y la colección de tareas.
- Se vincula el contexto de datos de la página a la clase principal.
- Se carga la lista de tareas desde Firebase al inicializar la página.

#### 4. Carga de Tareas:

- El método LoadTasks se encarga de obtener todas las tareas almacenadas en el nodo "Tareas" de Firebase.
- Se limpia la colección de tareas actual y se añaden las nuevas tareas obtenidas de Firebase a la colección observable.

#### 5. Agregar Tareas:

- El método AddTask se encarga de verificar que el campo de entrada no esté vacío.
- Crea un nuevo objeto de tarea (TaskItem) y lo publica en Firebase.
- Asigna el ID generado por Firebase a la nueva tarea y la añade a la colección observable.

#### 6. Actualizar Tareas:

- El método UpdateTask verifica que haya una tarea seleccionada y que el campo de entrada no esté vacío.
- Actualiza el nombre de la tarea seleccionada y refleja los cambios en Firebase.
- Recarga la lista de tareas para reflejar las actualizaciones.

#### 7. Eliminar Tareas:

- El método DeleteTask verifica que haya una tarea seleccionada.
- Elimina la tarea de Firebase usando su ID y la remueve de la colección observable.

#### 8. Selección de Tareas:

- El evento OnTaskSelected se dispara al seleccionar una tarea en la lista.
- Muestra el nombre de la tarea seleccionada en el campo de entrada para facilitar su actualización o eliminación.

```

using Firebase.Database;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.Formats.Tar;
using Firebase.Database.Query;

namespace ud04part2
{
    public partial class MainPage : ContentPage
    {
        private const string FirebaseUrl = "https://actfirebase317-default-rtdb.europe-west1.firebaseio.com/app/";
        private readonly FirebaseClient firebaseClient;

        public ObservableCollection<TaskItem> Tasks { get; set; }

        public MainPage()
        {
            InitializeComponent();

            firebaseClient = new FirebaseClient(FirebaseUrl);
            Tasks = new ObservableCollection<TaskItem>();
            BindingContext = this;

            LoadTasks();
        }

        private async void LoadTasks()
        {
            var firebaseTasks = await firebaseClient.Child("Tareas").OnceAsync<TaskItem>();
            Tasks.Clear();

            foreach (var item in firebaseTasks)
            {
                Tasks.Add(new TaskItem
                {
                    Id = item.Key,
                    NombreTarea = item.Object.NombreTarea
                });
            }
        }

        private async void AddTask(object sender, EventArgs e)
        {
            if (!string.IsNullOrEmpty(TaskEntry.Text))
            {
                var newTask = new TaskItem { NombreTarea = TaskEntry.Text };
                var response = await firebaseClient.Child("Tareas").PostAsync(newTask);

                newTask.Id = response.Key;
                Tasks.Add(newTask);
                TaskEntry.Text = string.Empty;
            }
        }

        private async void UpdateTask(object sender, EventArgs e)
        {
            if (TaskList.SelectedItem is TaskItem selectedTask && !
                string.IsNullOrEmpty(TaskEntry.Text))
            {
                selectedTask.NombreTarea = TaskEntry.Text;
                await firebaseClient.Child("Tareas").Child(selectedTask.Id).PutAsync(selectedTask);

                LoadTasks();
                TaskEntry.Text = string.Empty;
            }
        }

        private async void DeleteTask(object sender, EventArgs e)
        {
            if (TaskList.SelectedItem is TaskItem selectedTask)
            {
                await firebaseClient.Child("Tareas").Child(selectedTask.Id).DeleteAsync();
                Tasks.Remove(selectedTask);
            }
        }

        private void OnTaskSelected(object sender, SelectionChangedEventArgs e)
        {
            if (e.CurrentSelection.FirstOrDefault() is TaskItem selectedTask)
            {
                TaskEntry.Text = selectedTask.NombreTarea;
            }
        }
    }

    public class TaskItem
    {
        public string Id { get; set; }
        public string NombreTarea { get; set; }
    }
}

```

## PROBLEMA ENCONTRADO 🚧

Uno de los problemas principales que enfrenté fue al momento de implementar las operaciones con Firebase. Aunque logré conectar mi aplicación con la base de datos, no podía realizar consultas como acceder a nodos específicos con el método Child. El error surgía debido a que me faltaba importar la librería `Firebase.Database.Query`. Sin esta dependencia, el código no reconocía extensiones cruciales como Child o métodos como `PostAsync` y `PutAsync`, indispensables para interactuar con Firebase.

Esto provocó retrasos y una revisión exhaustiva del código y las dependencias. Sin embargo, después de identificar el problema, incorporar el `using Firebase.Database.Query`; resolvió completamente el inconveniente.

## SUGERENCIAS PARA HACER LA APP 💡

- **Revisar la documentación oficial de Firebase y las dependencias** utilizadas para asegurarse de que todas las librerías necesarias están correctamente incluidas.
- **Utilizar mensajes de error como una guía** para investigar posibles soluciones. El análisis de los errores que aparecían en el editor me ayudó a identificar rápidamente que faltaba un espacio de nombres.
- **Dividir el desarrollo en pasos claros y comprobables:** realizar pruebas después de cada nueva función ayuda a detectar problemas antes de avanzar demasiado.
- **Consultar ejemplos y guías**, especialmente para configuraciones como Firebase, que requieren pasos específicos para funcionar con .NET MAUI.

## CONCLUSIÓN

El desarrollo de esta aplicación fue una experiencia enriquecedora. Aunque enfrenté algunos problemas técnicos, como la falta de la librería `Firebase.Database.Query`, logré resolverlos y avanzar en mi comprensión de las herramientas y tecnologías involucradas. La integración de Firebase en una aplicación móvil es poderosa pero requiere atención a los detalles, especialmente en la configuración de dependencias y en la comprensión de cómo funcionan las consultas y operaciones con la base de datos.

El resultado es una aplicación funcional, bien estructurada y con una interfaz amigable y temática. Este proyecto no solo me ayudó a reforzar mis conocimientos en .NET MAUI, sino también a familiarizarme más con la gestión de datos en tiempo real utilizando Firebase.

## WEBGRAFÍA:

La documentación ha sido extraída de los recursos colocados por mi profesor Oscar Garcia en la plataforma educativa de Aules.

### Enlaces MUY útiles:

<https://firebase.google.com/docs/database>

<https://www.c-sharpcorner.com/article/firebase-realtime-database-with-c-sharp/>

<https://stackoverflow.com/questions/tagged/firebase>

<https://learn.microsoft.com/en-us/dotnet/maui/>

<https://learn.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.observablecollection>