Jorge Quintana González

Introducción

Vamos a hacer un trabajo en el módulo Diseño de interfaces (DI) en el cual tendremos que aprender cómo interactúa MAUI con la base de datos Firebase

objetivo

Tendremos que utilizar la herramienta Microsoft Visual Studio para completar los apartados del ejercicio.

Actividad

mainPage.xaml

Este código lo he hecho con la idea de que sea claro y funcional. He incluido un Entry para que el usuario pueda escribir el nombre de una nueva tarea. Me parece que esto es suficiente para comenzar, pero más adelante quiero implementar una validación para evitar que el usuario añada tareas vacías, ya que ahora mismo no lo controla.

Para mostrar las tareas, he utilizado un CollectionView porque es eficiente y permite manejar listas de forma más flexible. He diseñado un DataTemplate sencillo que muestra el Id y el Nombre de cada tarea en un diseño horizontal. Quiero que la información sea fácil de leer y que el diseño sea limpio, aunque podría añadir más detalles, como íconos o colores, para hacerlo más atractivo visualmente.

Finalmente, he añadido tres botones para las operaciones básicas: añadir, seleccionar y borrar tareas. Creo que con esto cubro las funciones principales de una lista de tareas, pero me he dado cuenta de que están un poco juntos y quizás deberían tener más espacio entre ellos o un diseño más visual con iconos.

```
<?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="UD02FirebaseJorge.MainPage">
    <ContentPage.Content>
        <StackLayout Padding="10">
            <Entry x:Name="NewTaskEntry" Placeholder="Escribe el nombre de la tarea" />
            <CollectionView x:Name="TaskListView" ItemsSource="{Binding Tasques}"
SelectionMode="Single">
                <CollectionView.ItemTemplate>
                    <DataTemplate>
                        <StackLayout Orientation="Horizontal" Padding="10">
                            <Label Text="{Binding Id}" WidthRequest="50" />
                            <Label Text="{Binding Nombre}" />
                        </StackLayout>
                    </DataTemplate>
                </CollectionView.ItemTemplate>
            </CollectionView>
            <HorizontalStackLayout>
                <Button Text="Añadir Tasca" Clicked="AddTaskButton_Clicked"/>
                <Button Text="Seleccionar Tasca" Clicked="SelectTaskButton_Clicked" />
                <Button Text="Borrar Tasca" Clicked="DeleteTaskButton_Clicked" />
            </HorizontalStackLayout>


        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

mainPage.xaml.cs

Este código lo he diseñado para gestionar una lista de tareas utilizando Firebase como base de datos en tiempo real. Me aseguré de que sea funcional y fácil de entender, pero también creo que tiene espacio para mejoras. Aquí comento algunos aspectos clave del código:

Para empezar, he creado una conexión a Firebase utilizando la URL de mi base de datos. Esto me permite acceder a las tareas almacenadas de manera remota. Me parece práctico cómo utilizo una colección observable (ObservableCollection<Tasca>) para que los datos se

actualicen automáticamente en la interfaz cada vez que cambian. Sin embargo, me gustaría optimizar la carga de datos si la lista de tareas crece mucho, ya que actualmente cargo todas las tareas cada vez.

El método GetNextId me asegura que cada nueva tarea tenga un ID único. Me gusta que manejo excepciones aquí, mostrando un mensaje claro si ocurre un error al obtener el próximo ID. Aun así, creo que podría mejorar el manejo de errores con una estructura más robusta, quizá registrando los errores en un log.

El botón para añadir tareas utiliza AddTaskButton_Clicked, que valida que el campo de entrada no esté vacío antes de crear una nueva tarea y enviarla a Firebase. Me parece útil que, tras añadir una tarea, el campo de entrada se limpie automáticamente. Sin embargo, un detalle que podría añadir es deshabilitar el botón mientras se realiza la operación, para evitar duplicados.

Para eliminar tareas, implementé DeleteTaskButton_Clicked, que elimina la tarea seleccionada tanto de Firebase como de la lista local. Me aseguré de manejar casos en los que no haya una tarea seleccionada o si la tarea no se encuentra en Firebase. Esto funciona bien, pero podría añadir un diálogo de confirmación antes de eliminar para evitar eliminaciones accidentales.

Finalmente, el botón para seleccionar tareas utiliza SelectTaskButton_Clicked, que muestra la información de la tarea seleccionada. Esto es simple pero efectivo. Quizá podría mejorar añadiendo la opción de editar la tarea directamente desde este mismo punto.

En general, creo que el código cubre las funcionalidades básicas que quería implementar, pero tengo ideas para refinarlo más adelante, como optimizar las consultas a Firebase o mejorar la experiencia del usuario con feedback más dinámico.

```
using Firebase.Database;
using Firebase.Database.Query;
using System;
using System.Collections.ObjectModel;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.Maui.Controls;

namespace UD02FirebaseJorge
{
    public partial class MainPage : ContentPage
    {
        private readonly FirebaseClient _firebaseClient;
        public ObservableCollection<Tasca> Tasques { get; set; }

        public MainPage()
        {
```

```
        /*http://fir-jorge-cac-73-default-rtdb.europe-west1.firebasedatabase.app
         * */
        InitializeComponent();
        _firebaseClient = new
FirebaseClient("https://fir-jorge-cac73-default-rtdb.europe-west1.firebasedatabase.app"); //
Reemplaza con tu URL de Firebase
        Tasques = new ObservableCollection<Tasca>();
        BindingContext = this;
        LoadTasks();
    }

    private async Task<int> GetNextId()
    {
        try
        {
            var tasks = await _firebaseClient.Child("Tasques").OnceAsync<Tasca>();
            return tasks.Any() ? tasks.Max(t => t.Object.Id) + 1 : 1;
        }
        catch (Exception ex)
        {
            await DisplayAlert("Error", $"Error obteniendo el próximo ID: {ex.Message}", "OK");
            return 1;
        }
    }

    private async void LoadTasks()
    {
        try
        {
            var firebaseTasks = await _firebaseClient.Child("Tasques").OnceAsync<Tasca>();
            Tasques.Clear();

            foreach (var task in firebaseTasks)
            {
                Tasques.Add(task.Object);
            }
        }
        catch (Exception ex)
        {
            await DisplayAlert("Error", $"Error cargando las tareas: {ex.Message}", "OK");
        }
    }

    private async void AddTaskButton_Clicked(object sender, EventArgs e)
    {
        try
        {
            if (!string.IsNullOrWhiteSpace(NewTaskEntry.Text))
```

```
            {
                var newTask = new Tasca(await GetNextId(), NewTaskEntry.Text);
                await _firebaseClient.Child("Tasques").PostAsync(newTask);
                Tasques.Add(newTask);
                NewTaskEntry.Text = string.Empty;
            }
            else
            {
                await DisplayAlert("Advertencia", "El campo de tarea no puede estar vacío.",
"OK");
            }
        }
        catch (Exception ex)
        {
            await DisplayAlert("Error", $"Error añadiendo la tarea: {ex.Message}", "OK");
        }
    }

    private async void DeleteTaskButton_Clicked(object sender, EventArgs e)
    {
        try
        {
            if (TaskListView.SelectedItem is Tasca selectedTask)
            {
                var firebaseTask = (await
_firebaseClient.Child("Tasques").OnceAsync<Tasca>())
                    .FirstOrDefault(t => t.Object.Id == selectedTask.Id);

                if (firebaseTask != null)
                {
                    await
_firebaseClient.Child("Tasques").Child(firebaseTask.Key).DeleteAsync();
                    Tasques.Remove(selectedTask);
                }
                else
                {
                    await DisplayAlert("Advertencia", "No se pudo encontrar la tarea
seleccionada en la base de datos.", "OK");
                }
            }
            else
            {
                await DisplayAlert("Advertencia", "Por favor selecciona una tarea para
eliminar.", "OK");
            }
        }
        catch (Exception ex)
        {
```

```
            await DisplayAlert("Error", $"Error eliminando la tarea: {ex.Message}", "OK");
        }
    }

    private void SelectTaskButton_Clicked(object sender, EventArgs e)
    {
        try
        {
            if (TaskListView.SelectedItem is Tasca selectedTask)
            {
                DisplayAlert("Tasca Seleccionada", $"ID: {selectedTask.Id}\nNombre:
{selectedTask.Nombre}", "OK");
            }
            else
            {
                DisplayAlert("Advertencia", "Por favor selecciona una tarea para ver.", "OK");
            }
        }
        catch (Exception ex)
        {
            DisplayAlert("Error", $"Error seleccionando la tarea: {ex.Message}", "OK");
        }
    }
  }

}
```

Tasca.cs

Este código lo he creado para definir la clase Tasca, que representa cada tarea en la aplicación. Creo que está bien organizado y cumple con su propósito, pero también hay algunos puntos en los que podría mejorar.

Primero, he implementado las propiedades Id y Nombre para identificar y describir cada tarea. Me gusta que el constructor de la clase permite inicializar estas propiedades fácilmente al crear una nueva tarea. Esto hace que el código sea más claro y directo al trabajar con objetos de tipo Tasca.

Además, he hecho que la clase implemente la interfaz INotifyPropertyChanged. Esto es importante porque me permite notificar a la interfaz de usuario cuando alguna propiedad de la clase cambia. Aunque en este caso las propiedades son simples y no tienen lógica adicional, me parece una buena práctica preparar la clase para futuras ampliaciones. Por ejemplo, si más adelante incluyo propiedades editables, esta funcionalidad ya estará lista.

El método OnPropertyChanged me asegura que los cambios en las propiedades se notifiquen correctamente. Para facilitar su uso, aproveché el atributo [CallerMemberName], que simplifica la llamada al método sin tener que especificar explícitamente el nombre de la propiedad. Esto hace el código más limpio y menos propenso a errores.

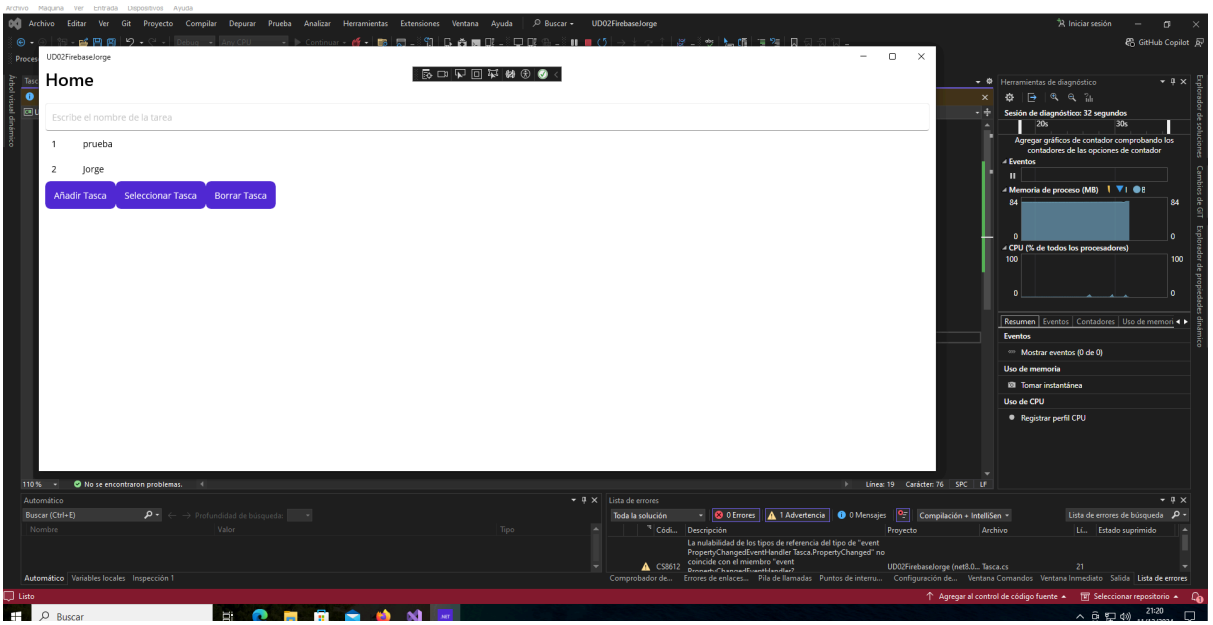
Aunque cumple con su función, me doy cuenta de que ahora mismo INotifyPropertyChanged no se utiliza en el código, ya que las propiedades son solo de lectura/escritura simples. Podría optimizarlo eliminando la implementación si estoy seguro de que no necesitaré esta funcionalidad, aunque prefiero dejarlo para futuras expansiones.

En resumen, este código es básico pero efectivo para representar una tarea en la aplicación. Tiene espacio para crecer, y creo que con algunas mejoras, como validar los valores asignados a las propiedades, podría ser aún más robusto.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

namespace UD02FirebaseJorge
{
    public partial class Tasca: INotifyPropertyChanged
    {
        public int Id { get; set; }
        public string Nombre { get; set; }

        public Tasca(int id, string nombre)
        {
            Id = id;
            Nombre = nombre;
        }
        public event PropertyChangedEventHandler PropertyChanged;
        protected void OnPropertyChanged([CallerMemberName] string name = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
    }
}
```


UD02FirebaseJorge
Home
Escribe el nombre de la tarea
1 prueba
2 Jorge
Añadir Tasca
Seleccionar Tasca
Borrar Tasca

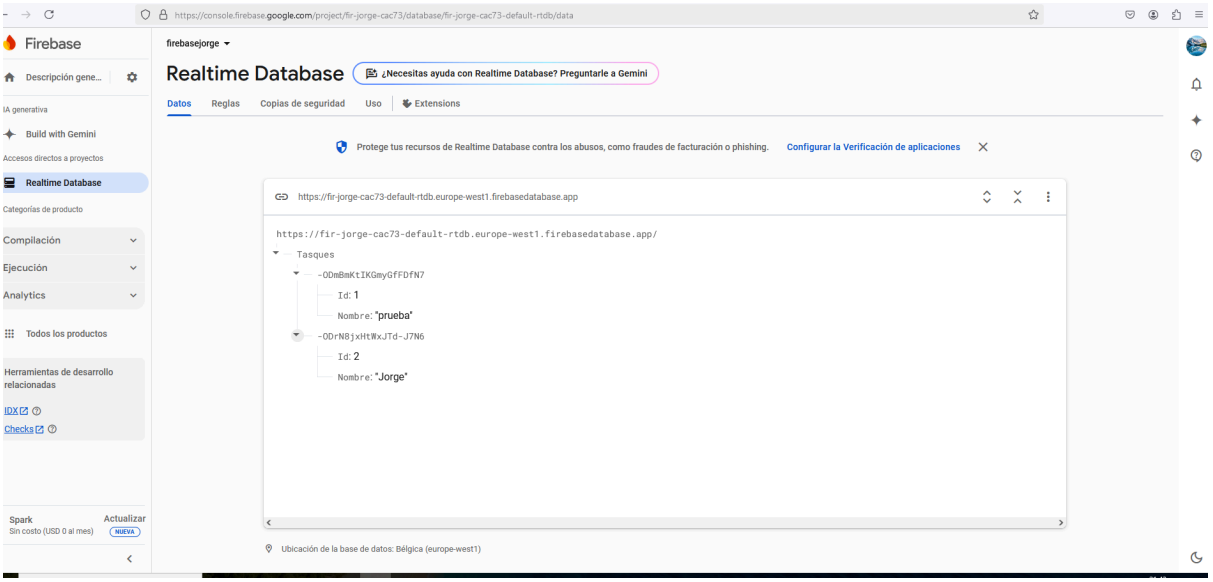
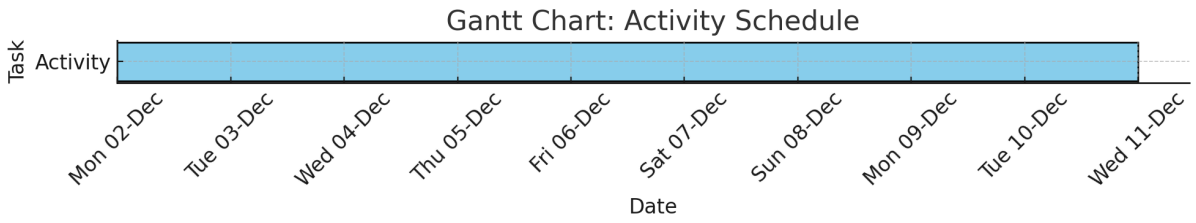

https://console.firebase.google.com/project/fir-jorge-cac73/database/fir-jorge-cac73-default-rtdb/data
Firebase
firebasejorge
Realtime Database
¿Necesitas ayuda con Realtime Database? Preguntarle a Gemini
Datos
Reglas
Copias de seguridad
Uso
Extensions
Protege tus recursos de Realtime Database contra los abusos, como fraudes de facturación o phishing.
Configurar la Verificación de aplicaciones
https://fir-jorge-cac73-default-rtdb.europe-west1.firebasedatabase.app
https://fir-jorge-cac73-default-rtdb.europe-west1.firebasedatabase.app/
Tasques
-ODmBmKtIKGmyGfFDfN7
Id: 1
Nombre: "prueba"
-ODrN8jxHtWxJTd-J7N6
Id: 2
Nombre: "Jorge"
Ubicación de la base de datos: Bélgica (europe-west1)

Diagrama de Gantt



Problemas

Conclusión

Creo que estos códigos forman una aplicación bastante útil para gestionar tareas, conectándose con Firebase como base de datos en tiempo real. Se pueden hacer cosas importantes como añadir, borrar y ver las tareas de forma sencilla. El código está bien organizado y es fácil de entender, así que no sería complicado ampliarlo en el futuro. Además, la clase Tasca con INotifyPropertyChanged está preparada para que la interfaz se actualice automáticamente si algo cambia, lo cual es una buena idea. Aun así, se podrían mejorar algunas cosas, como validar mejor los datos o hacer que las consultas a Firebase sean más rápidas si hay muchas tareas. En general, me parece un buen punto de partida para una aplicación más completa.