

# MEMORIA

## DESARROLLO DE INTERFACES



Alejandro Roberto Chiralt  
2ºDAM

## INDICE

INTRODUCCIÓN:.....	3
OBJETIVOS:.....	3
TAREAS A REALIZAR:.....	3
DESARROLLO :.....	4
ACTIVIDADES :.....	5
DIAGRAMA DE GANT:.....	12
PROBLEMAS:.....	12
CONCLUSIÓN:.....	12
WEBGRAFIA:.....	13

## INTRODUCCIÓN:

El presente proyecto desarrolla una aplicación multiplataforma utilizando .NET MAUI que permite gestionar información de empleados en una base de datos SQLite. La aplicación está diseñada para simplificar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre un listado de trabajadores. Este desarrollo integra la funcionalidad de almacenamiento local de datos con una interfaz de usuario intuitiva y adaptable a diferentes plataformas (Android, iOS, Windows, y MacOS).

## OBJETIVOS:

- **Principal:** Diseñar y desarrollar una aplicación capaz de gestionar información de empleados en una base de datos SQLite.
- Facilitar las operaciones de inserción, consulta, actualización y eliminación de registros desde una interfaz gráfica.
- Implementar una solución portable y funcional en múltiples plataformas.
- Asegurar la persistencia y accesibilidad de los datos almacenados.

## TAREAS A REALIZAR:

- **Diseño del sistema:**  
Definir la estructura de la base de datos, incluyendo las tablas y campos necesarios.  
Esquematizar la interfaz de usuario con los controles necesarios para las operaciones CRUD.
- **Implementación técnica:**  
Configurar el entorno de desarrollo para .NET MAUI y SQLite.  
Crear las clases y estructuras que representen los datos de los trabajadores.  
Desarrollar la lógica de negocio para interactuar con la base de datos.
- **Interfaz de usuario:**  
Implementar el diseño visual utilizando controles de .NET MAUI (como Entry, Button y CollectionView).  
Establecer el enlace de datos (data binding) entre la lógica del programa y la interfaz.
- **Pruebas y depuración:**

Realizar pruebas funcionales en diversas plataformas para asegurar la estabilidad y rendimiento.  
Depurar errores relacionados con la conexión a la base de datos y la interacción del usuario.

## DESARROLLO :

- **Estructura de la base de datos**

Se utiliza SQLite como sistema de almacenamiento local.

La base de datos contiene una tabla Trabajador con los siguientes campos:

- **id:** Identificador único, clave primaria.
- **nombre:** Nombre del trabajador.
- **apellidos:** Apellidos del trabajador.

- **Interfaz gráfica**

Los controles principales incluyen:

- **Entry:** Para la entrada de datos de nombre y apellidos.
- **Button:** Para ejecutar las acciones (Añadir, Actualizar, Eliminar).
- **CollectionView:** Para mostrar la lista de trabajadores almacenados.

- **Lógica de negocio**

**Creación de la base de datos:** Al inicializar la aplicación, se verifica la existencia de la base de datos y, en caso de que no exista, se crea junto con la tabla correspondiente.

**Operaciones CRUD:**

- **Crear:** Inserta un nuevo trabajador en la base de datos y lo añade al listado mostrado en la interfaz.
- **Leer:** Consulta todos los trabajadores almacenados y los muestra en un CollectionView.
- **Actualizar:** Modifica los datos de un trabajador seleccionado.
- **Eliminar:** Elimina un trabajador específico de la base de datos y de la lista visualizada.

- **Flujo de interacción**

Al cargar la aplicación, se inicializa el listado de trabajadores desde la base de datos.

Los botones permiten al usuario realizar acciones sobre los datos. Estas acciones están vinculadas a eventos que ejecutan las operaciones correspondientes en la base de datos.

Los cambios realizados en la base de datos se reflejan dinámicamente en la interfaz gracias al uso de ObservableCollection.

- **Pruebas**

Se realizaron pruebas en diversas plataformas para garantizar que la aplicación funcione correctamente.

Las pruebas incluyeron escenarios como:

- Inserción de datos vacíos.
- Actualización de trabajadores inexistentes.
- Manejo de errores en la conexión con la base de datos.

## ACTIVIDADES :

---

### Estructura del código

#### 1. Imports/Usings:

- Se importan bibliotecas necesarias para SQLite (Microsoft.Data.Sqlite y System.Data.SQLite) y para la construcción de la interfaz gráfica con .NET MAUI (Microsoft.Maui.Controls).

#### 2. Clase MainPage: Es la clase principal que contiene la lógica de la página principal de la aplicación.

---

### Propiedades

- **Propiedades de datos (Nombre, Apellidos, etc.):**

- Usadas para enlazar (binding) los datos de la interfaz gráfica con la lógica de la aplicación.
  - Incluyen:
    - **Nombre y Apellidos:** Para almacenar temporalmente los datos ingresados.
    - **NombreSeleccionado y ApellidoSeleccionado:** Para manejar los datos de un trabajador seleccionado en la interfaz.
    - **SelectedTrabajador:** Representa el trabajador seleccionado desde la interfaz.
    - **OcTrabajadores:** Es una colección observable (ObservableCollection) que contiene objetos de tipo Trabajador. Al agregar o modificar esta colección, la interfaz gráfica se actualiza automáticamente.
-

## Métodos

### 1. MainPage():

- Constructor de la clase.
- Inicializa la colección observable OcTrabajadores.
- Crea o abre la base de datos SQLite (empresa1.db), verifica si la tabla Trabajador existe, y la crea si no es así.
- Consulta todos los trabajadores existentes y los agrega a la colección OcTrabajadores para mostrarlos en la interfaz.

### 2. btnAñadirTrabajador\_Clicked:

- Se ejecuta al pulsar un botón para agregar un nuevo trabajador.
- Valida los campos de entrada (Nombre y Apellidos), guarda el trabajador en la base de datos y lo añade a la colección observable.

### 3. OnCollectionViewSelectionChanged:

- Maneja la selección de un trabajador en la interfaz.
- Actualiza los campos de entrada con la información del trabajador seleccionado.

### 4. btnBorrarTrabajador\_Clicked:

- Borra el trabajador seleccionado tanto de la base de datos como de la colección observable.

### 5. btnActualizar\_Clicked:

- Actualiza la información de un trabajador en la base de datos.
- Refleja los cambios en la colección observable y limpia los campos de entrada.

---

## Base de datos SQLite

### • Conexión:

- Se establece mediante SQLiteConnection y una cadena de conexión que define la ubicación de la base de datos (empresa1.db).

### • Operaciones principales:

- Crear tabla: CREATE TABLE IF NOT EXISTS Trabajador.
- Insertar registros: INSERT INTO Trabajador.
- Consultar registros: SELECT \* FROM Trabajador.
- Actualizar registros: UPDATE Trabajador.
- Eliminar registros: DELETE FROM Trabajador.

---

## Interacción con la interfaz gráfica

### • Binding Context:

- La propiedad BindingContext enlaza los datos de la clase (MainPage) con la interfaz gráfica.



- La colección OcTrabajadores está vinculada a un componente visual, como un CollectionView o ListView, lo que permite mostrar la lista de trabajadores en la interfaz.
- **Notificaciones de cambio:**
  - La implementación de OnPropertyChanged asegura que los cambios en propiedades como OcTrabajadores se reflejen automáticamente en la interfaz gráfica.

#### 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="AppAlmacenarBaseDeDatos.MainPage">
    <ScrollView>
        <VerticalStackLayout
            Padding="30,0"
            Spacing="25">
            <Label x:Name="tvTitulo" Text="BD Empresa"></Label>
            <Label x:Name="tvNombre" Text="Introduzca el nombre y apellidos del
trabajador"></Label>
            <Entry x:Name="etNombre" Placeholder="Nombre" ></Entry>
            <Entry x:Name="etApellidos" Placeholder="Apellidos" ></Entry>
            <Button x:Name="btnAñadirTrabajador" Text="Añadir Trabajador"
Clicked="btnAñadirTrabajador_Clicked"></Button>
            <Button x:Name="btnBorrarTrabajador" Text="Borrar Trabajador"
Clicked="btnBorrarTrabajador_Clicked"></Button>
            <Button x:Name="btnActualizar" Text="Actualizar Trabajador"
Clicked="btnActualizar_Clicked"></Button>
            <CollectionView
                ItemsSource="{Binding OcTrabajadores}"
                SelectionMode="Single"
                SelectionChanged="OnCollectionViewSelectionChanged"
                SelectedItem="{Binding SelectedTrabajador}"
                Margin="0,20,0,0">
                <CollectionView.ItemTemplate>
                    <DataTemplate>
                        <Frame BorderColor="LightGray" CornerRadius="10" Padding="10"
Margin="5">
                            <StackLayout>
                                <Label Text="{Binding NombreObjeto}"
FontAttributes="Bold" FontSize="18" />
                                <Label Text="{Binding ApellidosObjeto}" FontSize="14"
TextColor="Gray" />
                            </StackLayout>
                        </Frame>
                    </DataTemplate>
                </CollectionView.ItemTemplate>
            </CollectionView>
        </VerticalStackLayout>
    </ScrollView>
</ContentPage>
```

## Estructura del archivo

### 1. ContentPage:

- Representa la página principal de la aplicación.
- Atributos:
  - `xmlns` y `xmlns:x`: Espacios de nombres necesarios para el uso de XAML.
  - `x:Class`: Vincula esta página con la clase `MainPage` del archivo de código C#.

### 2. ScrollView:

- Permite que el contenido de la página sea desplazable si excede el tamaño de la pantalla.

### 3. VerticalStackLayout:

- Organiza los elementos verticalmente con un espacio (`Spacing="25"`) entre ellos.
- 

## Componentes principales

### Etiquetas y entradas de texto

#### 1. Label:

- `tvTitulo`: Muestra el título "BD Empresa".
- `tvNombre`: Informa al usuario sobre qué ingresar en los campos.

#### 2. Entry:

- Campos de texto donde el usuario puede ingresar el nombre (`etNombre`) y los apellidos (`etApellidos`) del trabajador.
- Atributo `Placeholder`: Texto que se muestra como sugerencia cuando el campo está vacío.

### Botones

- Tres botones que activan eventos en la lógica C#:
  1. `btnAñadirTrabajador`: Ejecuta el evento `btnAñadirTrabajador_Clicked` para agregar un trabajador.
  2. `btnBorrarTrabajador`: Ejecuta `btnBorrarTrabajador_Clicked` para eliminar un trabajador seleccionado.
  3. `btnActualizar`: Ejecuta `btnActualizar_Clicked` para modificar la información del trabajador seleccionado.

### CollectionView:

- Muestra una lista de trabajadores almacenados en la colección observable (`OcTrabajadores`).
- Atributos destacados:
  - `ItemsSource="{Binding OcTrabajadores}"`: Enlaza la colección con el contenido del `CollectionView`.
  - `SelectionMode="Single"`: Permite seleccionar un solo trabajador.



- `SelectionChanged="OnCollectionViewSelectionChanged"`: Llama al evento que maneja los cambios de selección.
  - `SelectedItem="{Binding SelectedTrabajador}"`: Vincula el trabajador seleccionado con la propiedad `SelectedTrabajador` en el archivo C#.
  - **Plantilla de elementos (ItemTemplate):**
    - Cada trabajador se representa mediante un `Frame` que contiene:
      - `Label` para el nombre (`NombreObjeto`).
      - `Label` para los apellidos (`ApellidosObjeto`).
    - Diseño atractivo con:
      - Borde gris claro (`BorderColor="LightGray"`).
      - Esquinas redondeadas (`CornerRadius="10"`).
      - Espaciado interno (`Padding="10"`) y margen externo (`Margin="5"`).
- 

## Vinculación entre XAML y C#

### 1. Eventos:

- Los eventos de botones y selección están conectados a métodos en `MainPage.xaml.cs`.
- Ejemplo: Al pulsar `btnAñadirTrabajador`, se ejecuta el método `btnAñadirTrabajador_Clicked`.

### 2. Binding Context:

- El archivo XAML utiliza el contexto de datos (`BindingContext`) configurado en `MainPage.xaml.cs` para interactuar con propiedades como `OcTrabajadores` y `SelectedTrabajador`.

### 3. Actualización de la interfaz:

- Cuando `OcTrabajadores` cambia, el `CollectionView` se actualiza automáticamente gracias al soporte de `ObservableCollection`.
- 

## Flujo de uso

### 1. El usuario:

- Ingresa el nombre y apellidos en los campos de texto.
- Usa los botones para agregar, eliminar o actualizar trabajadores.

### 2. Los cambios en los datos:

- Se reflejan en el `CollectionView`, mostrando en tiempo real la lista actualizada de trabajadores.

## MainPage.xaml.cs

```
using Microsoft.Data.Sqlite;
using Microsoft.Maui.Controls;
using System.Collections.ObjectModel;
using System.Data.SQLite;
using System.Reflection.PortableExecutable;
using System.Xml.Linq;
namespace AppAlmacenarBaseDeDatos
{
    public partial class MainPage : ContentPage
    {
        private string _nombre;
        public string Nombre
        {
            get
            {
                return this._nombre;
            }
            set
            {
                if ((this._nombre != value))
                {
                    this._nombre = value;
                }
            }
        }

        private string _apellidos;
        public string Apellidos
        {
            get
            {
                return this._apellidos;
            }
            set
            {
                if (this._apellidos != value)
                {
                    this._apellidos = value;
                }
            }
        }
    }

    private string _nombreSeleccionado;
    public string NombreSeleccionado
    {
        get
        {
            return this._nombreSeleccionado;
        }
        set
        {
            if (this._nombreSeleccionado != value)
            {
                this._nombreSeleccionado = value;
            }
        }
    }

    private string _apellidoSeleccionado;
    public string ApellidoSeleccionado
    {
        get
        {
            return this._apellidoSeleccionado;
        }
    }
}
```

```

        set
        {
            if (this._apellidoSeleccionado != value)
            {
                this._apellidoSeleccionado = value;
            }
        }
    }
}

private Trabajador _selectedTrabajador;
public Trabajador SelectedTrabajador
{
    get
    {
        return this._selectedTrabajador;
    }
    set
    {
        _selectedTrabajador = value;
        OnPropertyChanged();
    }
}

private ObservableCollection<Trabajador> _ocTrabajadores;
public ObservableCollection<Trabajador> OcTrabajadores
{
    get { return _ocTrabajadores; }
    set
    {
        _ocTrabajadores = value;
        OnPropertyChanged();
    }
}

public MainPage()
{
    InitializeComponent();
    OcTrabajadores = new ObservableCollection<Trabajador>();
    //CREAR BD
    string rutaDirectorioApp = System.AppContext.BaseDirectory;
    DirectoryInfo directorioApp = new DirectoryInfo(rutaDirectorioApp);
    directorioApp = directorioApp.Parent.Parent.Parent.Parent.Parent.Parent;
    string databasePath = Path.Combine(directorioApp.FullName, "empresa1.db");
    string connectionString = $"Data Source={databasePath};Version=3;";
    try
    {
        if (!File.Exists(databasePath))
        {
            using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
            {
                connection.Open();
                DisplayAlert("Éxito", "Conexión exitosa con la base de datos.", "OK");
                string queryCrearTabla = @"
                CREATE TABLE IF NOT EXISTS Trabajador (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                nombre TEXT NOT NULL,
                apellidos TEXT NOT NULL
                );";
                using (SQLiteCommand command = new SQLiteCommand(queryCrearTabla,
connection))
                {
                    command.ExecuteNonQuery();
                }
                connection.Close();
            }
        }
    }
}

```

```

    }
    catch (Exception ex)
    {
        DisplayAlert("Error", $"Error al insertar en la base de datos: {ex.Message}",
"OK");
    }
    //CONSULTA
    using (SQLiteConnection connection = new SQLiteConnection(connectionString))
    {
        connection.Open();
        // Creamos la consulta y la ejecutamos
        string sql = "SELECT * FROM Trabajador";
        SQLiteCommand command = new SQLiteCommand(sql, connection);
        SQLiteDataReader reader = command.ExecuteReader();
        // Recorremos los registros devueltos del SELECT
        while (reader.Read())
        {
            string _nombre = reader.GetString(1);
            string _apellidos = reader.GetString(2);
            // Creamos un objeto Trabajador y lo añadimos al Observable Collection
            Trabajador trabajador = new Trabajador
            {
                NombreObjeto = _nombre,
                ApellidosObjeto = _apellidos,
            };
            // Añadimos el trabajador al Observable Collection
            OcTrabajadores.Add(trabajador);
        }
        reader.Close();
        connection.Close();
    }
    BindingContext = this;
}

private async void btnAñadirTrabajador_Clicked(object sender, EventArgs e)
{
    // Capturar los datos de entrada
    _nombre = etNombre.Text;
    _apellidos = etApellidos.Text;
    // Validar que no sean nulos o vacíos
    if (string.IsNullOrEmpty(_nombre) || string.IsNullOrEmpty(_apellidos))
    {
        Console.WriteLine("El nombre o los apellidos están vacíos.");
        DisplayAlert("Fail", "Nombre o apellidos vacio", "OK");
        return;
    }
    // Crear un nuevo trabajador
    Trabajador trabajador = new Trabajador
    {
        NombreObjeto = _nombre,
        ApellidosObjeto = _apellidos,
    };

    // Notificar cambio en la colección (para bindings en la UI)
    OnPropertyChanged(nameof(OcTrabajadores));
    // Guardar en la base de datos
    string rutaDirectorioApp = System.AppContext.BaseDirectory;
    DirectoryInfo directorioApp = new DirectoryInfo(rutaDirectorioApp);
    directorioApp = directorioApp.Parent.Parent.Parent.Parent.Parent.Parent;
    string databasePath = Path.Combine(directorioApp.FullName, "empresa1.db");
    string connectionString = $"Data Source={databasePath};Version=3;";

    try
    {

```

```

using (SQLiteConnection connection = new SQLiteConnection(connectionString))
{
    connection.Open();
    DisplayAlert("Éxito", "Conexión exitosa con la base de datos.", "OK");
    string queryInsertar = "INSERT INTO Trabajador (nombre, apellidos) VALUES (@nombre, @apellidos)";
    using (SQLiteCommand command = new SQLiteCommand(queryInsertar, connection))
    {
        command.Parameters.AddWithValue("@nombre", _nombre);
        command.Parameters.AddWithValue("@apellidos", _apellidos);
        int filasAfectadas = command.ExecuteNonQuery();
        if (filasAfectadas > 0)
        {
            DisplayAlert("Trabajador añadido correctamente",
                $"Nombre: {_nombre}\nApellidos: {_apellidos}",
                "OK");
        }
        else
        {
            DisplayAlert("Error", "No se pudo insertar el trabajador.", "OK");
        }
    }
    connection.Close();
}
}
catch (Exception ex)
{
    DisplayAlert("Error", $"Error al insertar en la base de datos: {ex.Message}",
"OK");
}
// Añadir a la colección
OcTrabajadores.Add(new Trabajador { NombreObjeto = _nombre, ApellidosObjeto =
_apellidos });

// Limpiar los campos de entrada
etNombre.Text = string.Empty;
etApellidos.Text = string.Empty;
}

private void OnCollectionViewSelectionChanged(object sender, SelectionChangedEventArgs
e)
{
    // Obtener el elemento seleccionado
    var selectedTrabajador = e.CurrentSelection.FirstOrDefault() as Trabajador;
    if (selectedTrabajador != null)
    {
        // Realiza alguna acción con el trabajador seleccionado
        DisplayAlert("Trabajador Seleccionado",
            $"Nombre: {selectedTrabajador.NombreObjeto}\nApellidos:
{selectedTrabajador.ApellidosObjeto}",
            "OK");
        etNombre.Text = selectedTrabajador.NombreObjeto;
        etApellidos.Text = selectedTrabajador.ApellidosObjeto;
        _nombreSeleccionado = selectedTrabajador.NombreObjeto;
        _apellidoSeleccionado = selectedTrabajador.ApellidosObjeto;
        OnPropertyChanged(nameof(OcTrabajadores));
    }
}

private void btnBorrarTrabajador_Clicked(object sender, EventArgs e)
{
    // Borrar en la base de datos
    string rutaDirectorioApp = System.AppContext.BaseDirectory;
    DirectoryInfo directorioApp = new DirectoryInfo(rutaDirectorioApp);

```

```

directorioApp = directorioApp.Parent.Parent.Parent.Parent.Parent.Parent;
string databasePath = Path.Combine(directorioApp.FullName, "empresa1.db");
string connectionString = $"Data Source={databasePath};Version=3;";
try
{
    using (SQLiteConnection connection = new SQLiteConnection(connectionString))
    {
        connection.Open();
        DisplayAlert("Éxito", "Conexión exitosa con la base de datos.", "OK");
        string queryInsertar = "DELETE FROM Trabajador WHERE nombre = @nombre AND
apellidos = @apellidos";
        using (SQLiteCommand command = new SQLiteCommand(queryInsertar,
connection))
        {
            command.Parameters.AddWithValue("@nombre",
SelectedTrabajador.NombreObjeto);
            command.Parameters.AddWithValue("@apellidos",
SelectedTrabajador.ApellidosObjeto);
            int filasAfectadas = command.ExecuteNonQuery();
            if (filasAfectadas > 0)
            {
                DisplayAlert("Trabajador borrado correctamente",
$"Nombre: {SelectedTrabajador.NombreObjeto}\nApellidos:
{SelectedTrabajador.ApellidosObjeto}",
"OK");
            }
            else
            {
                DisplayAlert("Error", "No se pudo borrar el trabajador.", "OK");
            }
        }
        connection.Close();
    }
}
catch (Exception ex)
{
    DisplayAlert("Error", $"Error al borrar de la base de datos: {ex.Message}",
"OK");
}
OcTrabajadores.Remove(SelectedTrabajador);
// Limpiar los campos de entrada
etNombre.Text = string.Empty;
etApellidos.Text = string.Empty;
}

private void btnActualizar_Clicked(object sender, EventArgs e)
{
    // Actualizar en la base de datos
    string rutaDirectorioApp = System.AppContext.BaseDirectory;
    DirectoryInfo directorioApp = new DirectoryInfo(rutaDirectorioApp);
    directorioApp = directorioApp.Parent.Parent.Parent.Parent.Parent.Parent;
    string databasePath = Path.Combine(directorioApp.FullName, "empresa1.db");
    string connectionString = $"Data Source={databasePath};Version=3;";
    try
    {
        using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
        {
            connection.Open();
            string queryActualizar = "UPDATE Trabajador SET nombre = @nombre,
apellidos = @apellidos WHERE nombre = @nombreSelect AND apellidos = @apellidoSelect\r\
n";

            // Capturar los datos de entrada
            _nombre = etNombre.Text;
            _apellidos = etApellidos.Text;

```



```

        using (SQLiteCommand command = new SQLiteCommand(queryActualizar,
connection))
        {
            command.Parameters.AddWithValue("@nombre", _nombre);
            command.Parameters.AddWithValue("@apellidos", _apellidos);
            command.Parameters.AddWithValue("@nombreSelect",
_nombreSeleccionado);
            command.Parameters.AddWithValue("@apellidoSelect",
_apellidoSeleccionado);
            // Añadir a la colección
            OcTrabajadores.Add(new Trabajador { NombreObjeto = _nombre,
ApellidosObjeto = _apellidos });
            OnPropertyChanged(nameof(OcTrabajadores));
            int filasAfectadas = command.ExecuteNonQuery();
            if (filasAfectadas > 0)
            {
                DisplayAlert("Trabajador actualizado correctamente",
                    $"Nombre: {_nombre}\nApellidos: {_apellidos}",
                    "OK");
            }
            else
            {
                DisplayAlert("Error", "No se pudo actualizado el
trabajador.", "OK");
            }
        }
        connection.Close();
    }
}
catch (Exception ex)
{
    DisplayAlert("Error", $"Error al insertar en la base de datos:
{ex.Message}", "OK");
}
// Limpiar los campos de entrada
etNombre.Text = string.Empty;
etApellidos.Text = string.Empty;
BindingContext = this;
}
}
}

```

## Propiedades

### 1. NombreObjeto:

- Representa el nombre de un trabajador.
- Es de tipo `string` y tiene un `getter` y un `setter` automáticos.

### 2. ApellidosObjeto:

- Representa los apellidos de un trabajador.
- También tiene un `getter` y un `setter` automáticos.

---

## Interfaz `INotifyPropertyChanged`

- Implementa esta interfaz para notificar cambios en las propiedades, lo que es útil cuando las propiedades están vinculadas a la interfaz de usuario (**Binding** en .NET MAUI).
- **Evento `PropertyChanged`:**
  - Se dispara cuando una propiedad de la clase cambia.
  - Vinculado a cualquier control en la UI que dependa de la propiedad.
- **Método `OnPropertyChanged`:**
  - Se utiliza para invocar el evento `PropertyChanged`.
  - Parámetro `[CallerMemberName]`:
    - Proporciona automáticamente el nombre de la propiedad que invoca este método, por lo que no necesitas pasar manualmente el nombre.

## Trabajadores.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;
using static System.Runtime.InteropServices.JavaScript.JSType;
namespace AppAlmacenarBaseDeDatos
{
    public class Trabajador : INotifyPropertyChanged
    {
        public string NombreObjeto { get; set; }
        public string ApellidosObjeto { get; set; }
        public event PropertyChangedEventHandler PropertyChanged;
        protected void OnPropertyChanged([CallerMemberName] string name = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
        public override string ToString()
        {
            return $"Nombre: {NombreObjeto}, Apellidos: {ApellidosObjeto}";
        }
    }
}
```

# FUNCIONAMIENTO:

Introducimos los parametros de Nombre y Apellido, para añadir al trabajador, y le damos al botón de "Añadir Trabajador".

AppAlmacenarBaseDeDatos

## Home

BD Empresa

Introduzca el nombre y apellidos del trabajador

Añadir Trabajador

Borrar Trabajador

Actualizar Trabajador

Vemos como se ha actualizado el CollectionView y podemos ver y seleccionar el Trabajador "Alejandro Roberto"

AppAlmacenarBaseDeDatos

## Home

BD Empresa

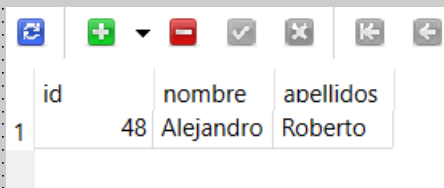
Añadir Trabajador

Borrar Trabajador

Actualizar Trabajador

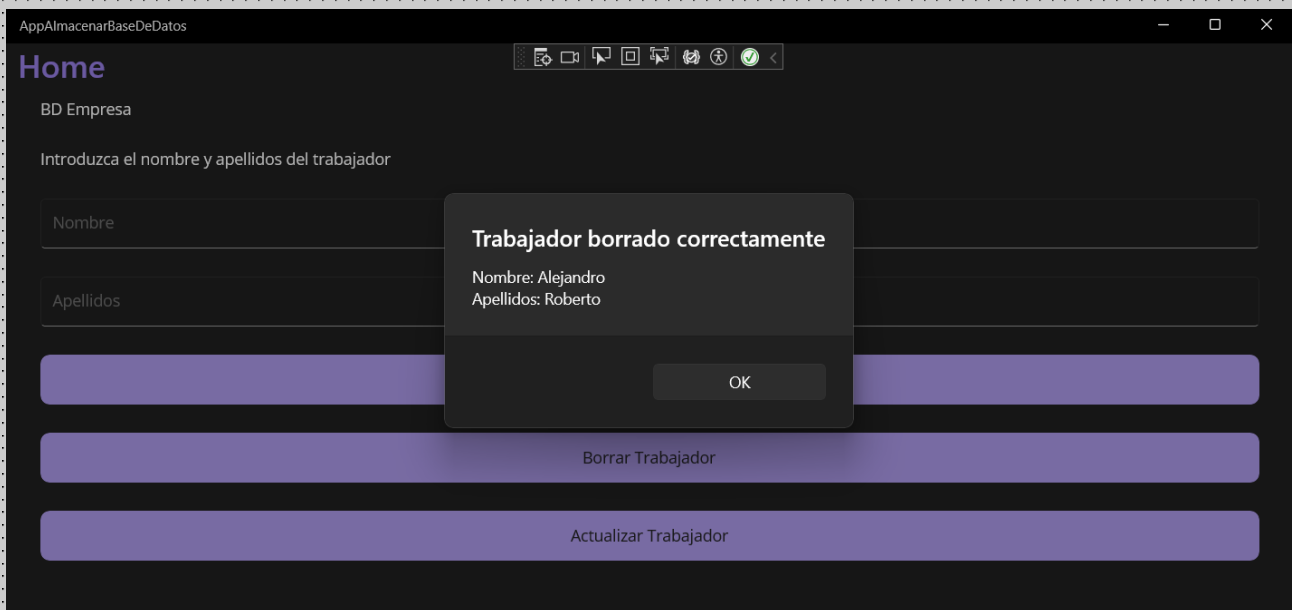
**Alejandro**  
Roberto

Aquí estamos viendo al base de datos, para ver como se ha actualizado y se ha añadido el Trabajador.



id	nombre	apellidos
1	48 Alejandro	Roberto

A continuación, vamos a borrar el Trabajador "Alejandro Roberto", seleccionandolo abajo y dandole al boton de "Borrar Trabajador"



AppAlmacenarBaseDeDatos

## Home

BD Empresa

Introduzca el nombre y apellidos del trabajador

Nombre

Apellidos

**Trabajador borrado correctamente**

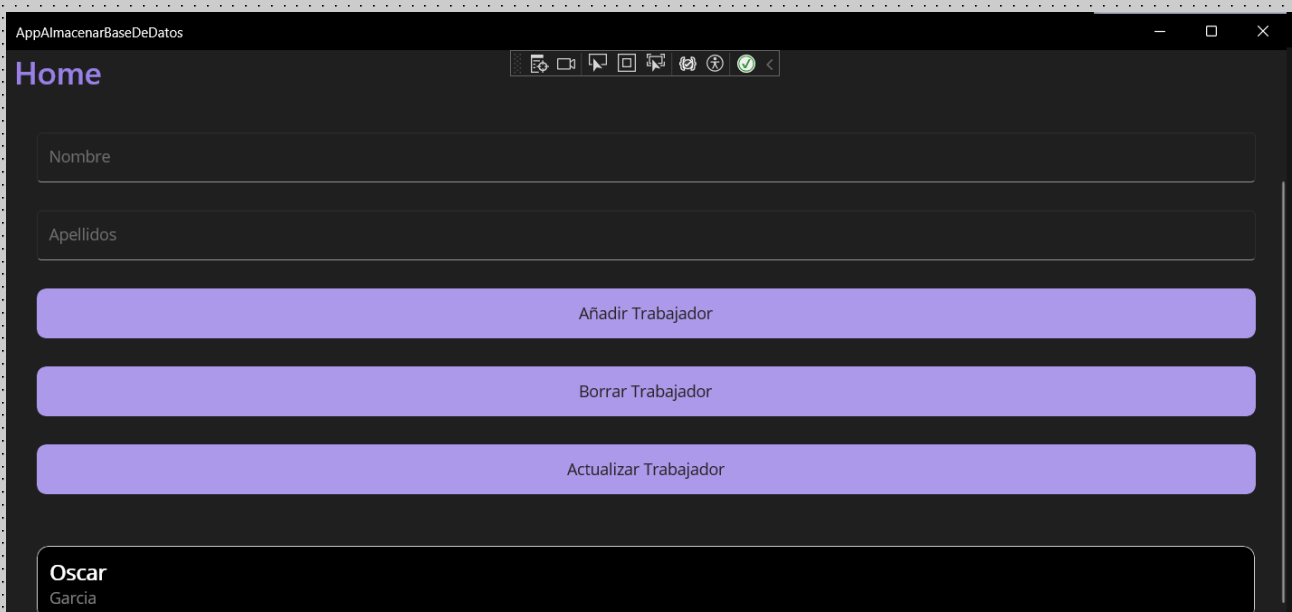
Nombre: Alejandro  
Apellidos: Roberto

OK

Borrar Trabajador

Actualizar Trabajador

Ahora hemos añadido un nuevo Trabajador, para poder hacer el resto de opciones con el.



AppAlmacenarBaseDeDatos

## Home

Nombre

Apellidos

Añadir Trabajador

Borrar Trabajador

Actualizar Trabajador

**Oscar**  
García

Vemos como se ha actualizado la base de datos.

	id	nombre	apellidos
1	49	Oscar	Garcia

Ahora hemos selccionado el Trabajador y cambiamos los datos que queramos actualizar, a continuación le damos al boton de "Actualizar Trabajador".

The screenshot shows a mobile application window titled "AppAlmacenarBaseDeDatos". The interface has a dark theme. At the top, there's a "Home" header. Below it, there are input fields for "Nombre" and "Apellidos". A modal dialog is displayed in the center with the title "Trabajador actualizado correctamente" and the text "Nombre: Oscar" and "Apellidos: Profesor". Below the dialog, there's a large blue button labeled "Actualizar Trabajador". At the bottom, there's a card displaying the updated worker's information: "Oscar Garcia".

Y vemos como se ha actualizado el Trabajador en la base de datos , con los datos introducidos previamente.

	id	nombre	apellidos
1	49	Oscar	Profesor

Aquí también vemos como se ha actualizado en el ObservableCollection.

AppAlmacenarBaseDeDatos

## Home

BD Empresa

Introduzca el nombre y apellidos del trabajador

Nombre

Apellidos

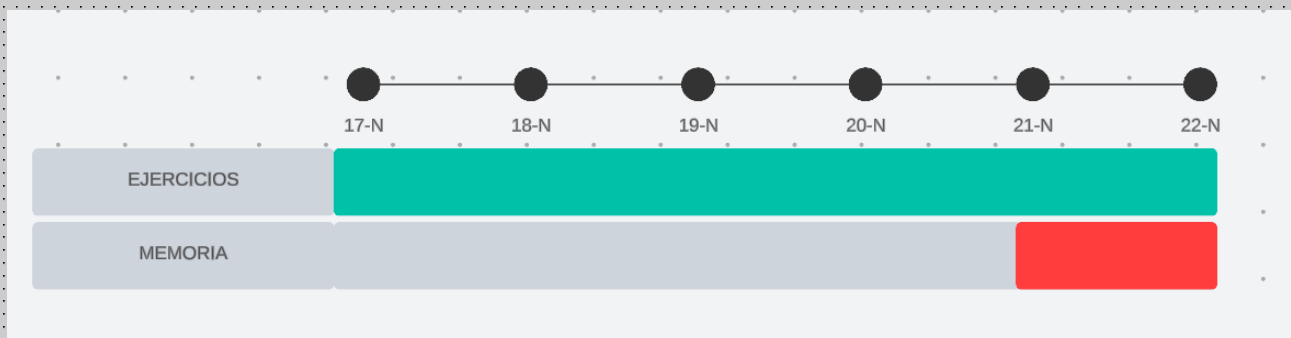
Añadir Trabajador

Borrar Trabajador

Actualizar Trabajador

Oscar  
Profesor

## DIAGRAMA DE GANT:



## PROBLEMAS:

Los mayores problemas que he tenido ha sido a la hora de comprender como funciona MAUI , y el como almacenar y hacer un setter de los datos. A la hora de actualizar, se actualiza de la base de datos, pero del ObservableCollection.



## CONCLUSIÓN:

El entorno de desarrollo de Visual Studio va bastante mal, y no es muy intuitivo.

## WEBGRAFIA:

<https://learn.microsoft.com/es-es/dotnet/maui/get-started/first-app?view=net-maui-8.0&tabs=vswin&pivots=devices-android>