CRUD - SQLite en Xamarin.Forms Proyecto Integrador Checklist

Ríos Rodríguez María Fernanda
Aguilar Ivan Camilo
Roncancio Roldan Alejandro

Universidad San Buenaventura Tecnología de Desarrollo de Software
Profesora Sandra Sánchez
Noviembre del 2025

CRUD Proyecto Integrador Checklist

1. Descripción General del proyecto:

La aplicación **"Checklist"** es una app móvil desarrollada en **Xamarin.Forms** que permite gestionar un registro de profesores utilizando una base de datos **SQLite**.
El sistema realiza operaciones CRUD completas:

- **Crear** nuevos profesores
- **Listar** todos los registros guardados
- **Actualizar** la información existente
- **Eliminar** un profesor seleccionado

La app funciona completamente **offline**, almacenando los datos en el dispositivo móvil (Android o emulador) mediante **SQLite** y la librería sqlite-net-pcl.

2. Modelo de datos:

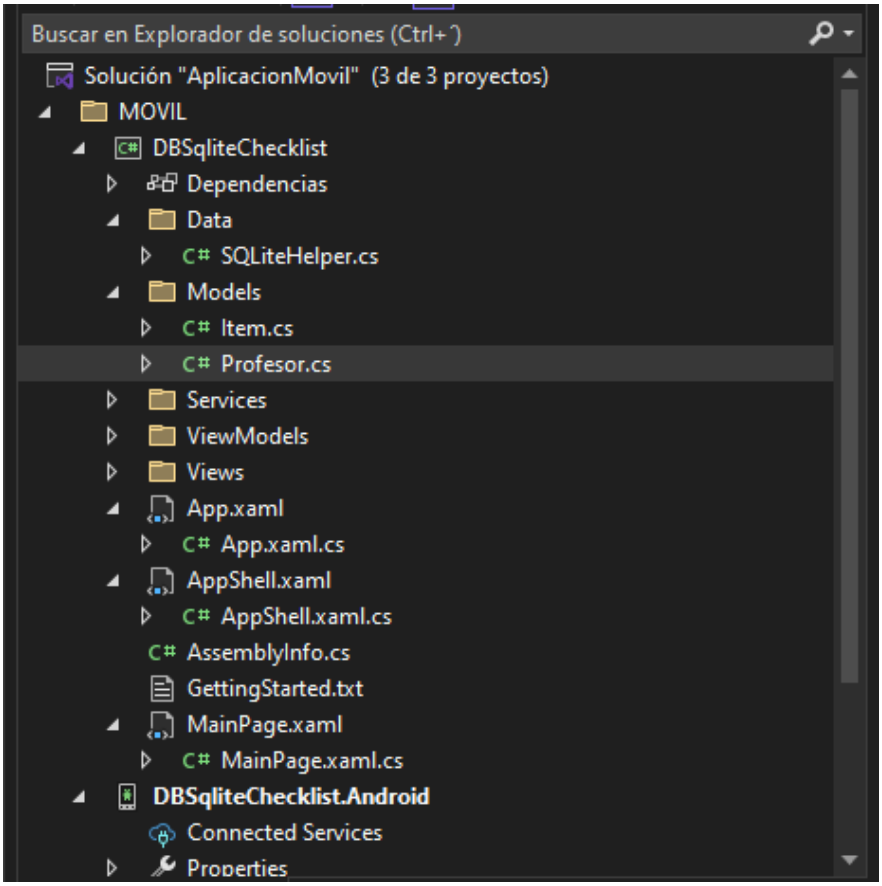
- **Entidad principal:** Profesor (definida en *Models/Profesor.cs*)

Campo	Tipo de dato	Descripción
IdProfesor	`string` (PK)	Identificador único del profesor
Nombre	`string`	Nombre del profesor
Apellido	`string`	Apellido del profesor

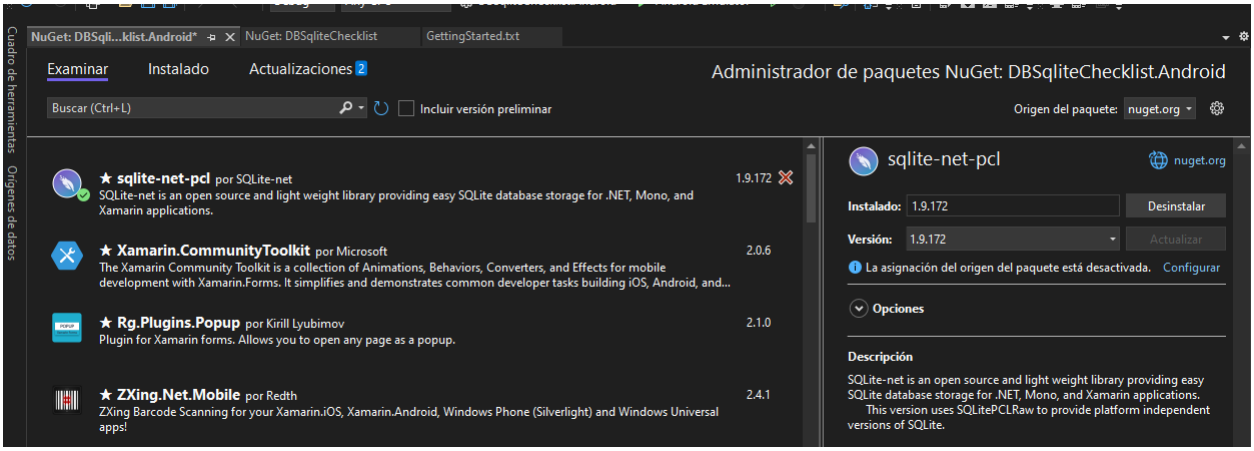
3. Estructura del proyecto:

DBSqliteChecklist

- ├ Data
  - │ └ SQLiteHelper.cs ← *conexión y métodos SQLite (CRUD)*
- ├ Models
  - │ └ Profesor.cs ← *modelo de datos*
- ├ Pages (o raíz)
  - │ ├ MainPage.xaml ← *interfaz visual principal*
  - │ └ MainPage.xaml.cs ← *lógica de interacción (CRUD)*
- ├ App.xaml ← *configuración global*
- └ App.xaml.cs ← *inicialización y conexión BD*



- Seguidamente oprimir botón derecho sobre la solución DBSqlite y elige del menú desplegable Administrar paquetes NuGet para solución (Manage NuGet Packages For Solucion). De la ventana ubícate en buscar y digita sqlite net pcl luego seleccionar el que se muestra en la figura y descargarlo.



1. **Explicación código:**

   ➢ **SQLiteHelper.cs:**

   Implementa la conexión con la base de datos y los métodos asíncronos que gestionan los datos. Utiliza `SQLiteAsyncConnection` para realizar operaciones sin bloquear la interfaz de usuario lo cual generaremos la creación, actualización, mostrar y eeliminar profesores.

   Métodos principales:

   - GuardarProfesorAsync(Profesor prof) → Inserta o reemplaza un registro.
   - ObtenerProfesoresAsync() → Retorna una lista de profesores.
   - ActualizarProfesorAsync(Profesor prof) → Modifica un registro existente.
   - EliminarProfesorAsync(Profesor prof) → Elimina un registro de la tabla.

```
using System.Collections.Generic;
using System.Threading.Tasks;
using SQLite;
using DBSqliteChecklist.Models;

namespace DBSqliteChecklist.Data
{
    4 referencias
    public class SQLiteHelper
    {
        readonly SQLiteAsyncConnection db;

        1 referencia
        public SQLiteHelper(string dbPath)
        {
            db = new SQLiteAsyncConnection(dbPath);
            db.CreateTableAsync<Profesor>().Wait();
        }

        // Guardar o reemplazar
        1 referencia
        public Task<int> GuardarProfesorAsync(Profesor prof)
        {
            if (!string.IsNullOrEmpty(prof.IdProfesor))
                return db.InsertOrReplaceAsync(prof);
            else
                return db.InsertAsync(prof);
        }

        // Obtener todos los profesores
        1 referencia
        public Task<List<Profesor>> ObtenerProfesoresAsync()
        {
            return db.Table<Profesor>().ToListAsync();
        }

        // Actualizar profesor
        1 referencia
        public Task<int> ActualizarProfesorAsync(Profesor prof)
        {
            return db.UpdateAsync(prof);
        }

        // ✅ Nuevo método: eliminar profesor
        1 referencia
        public Task<int> EliminarProfesorAsync(Profesor prof)
        {
            return db.DeleteAsync(prof);
        }
```

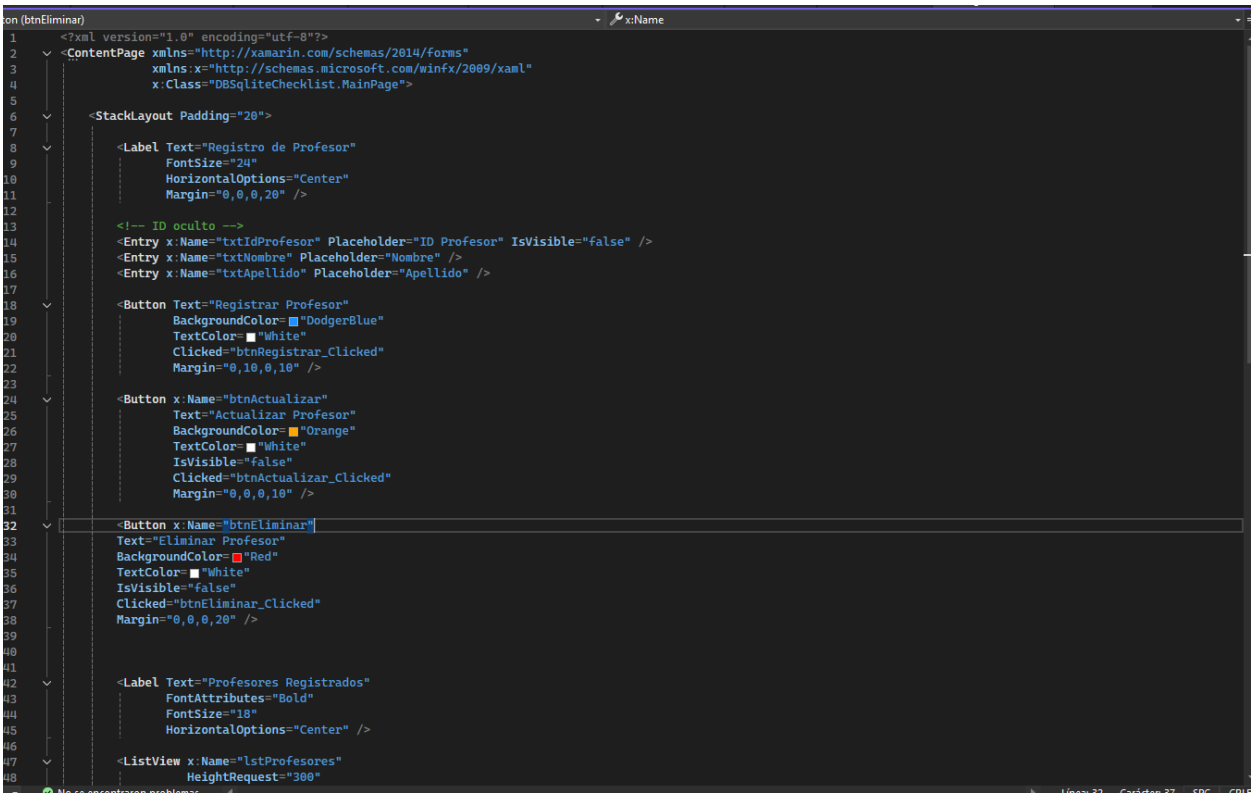
- **Profesor.cs:**

  Contiene el **modelo de datos** que define la entidad Profesor. Cada propiedad representa una columna en la tabla SQLite. Incluye el atributo [PrimaryKey] para indicar que IdProfesor es la clave principal.

```
DBSqliteChecklist                                   DBSqliteChecklist.Models.Profesor
1   using SQLite;
2
3   namespace DBSqliteChecklist.Models
4   {
        13 referencias
5       public class Profesor
6       {
7           [PrimaryKey]
            5 referencias
8           public string IdProfesor { get; set; }
            4 referencias
9           public string Nombre { get; set; }
            4 referencias
10          public string Apellido { get; set; }
11      }
12  }
13
```

- **MainPage.xaml:**

  Define la **interfaz visual** del formulario principal utilizando XAML. Contiene los campos de texto (`Entry`) y los botones para las acciones CRUD.



```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DBSqliteChecklist.MainPage">

    <StackLayout Padding="20">

        <Label Text="Registro de Profesor"
               FontSize="24"
               HorizontalOptions="Center"
               Margin="0,0,0,20" />

        <!-- ID oculto -->
        <Entry x:Name="txtIdProfesor" Placeholder="ID Profesor" IsVisible="false" />
        <Entry x:Name="txtNombre" Placeholder="Nombre" />
        <Entry x:Name="txtApellido" Placeholder="Apellido" />

        <Button Text="Registrar Profesor"
                BackgroundColor="DodgerBlue"
                TextColor="White"
                Clicked="btnRegistrar_Clicked"
                Margin="0,10,0,10" />

        <Button x:Name="btnActualizar"
                Text="Actualizar Profesor"
                BackgroundColor="Orange"
                TextColor="White"
                IsVisible="false"
                Clicked="btnActualizar_Clicked"
                Margin="0,0,0,10" />

        <Button x:Name="btnEliminar"
        Text="Eliminar Profesor"
        BackgroundColor="Red"
        TextColor="White"
        IsVisible="false"
        Clicked="btnEliminar_Clicked"
        Margin="0,0,0,20" />



        <Label Text="Profesores Registrados"
               FontAttributes="Bold"
               FontSize="18"
               HorizontalOptions="Center" />

        <ListView x:Name="lstProfesores"
                  HeightRequest="300"
```

- **MainPage.xaml.cs:**

  Contiene la **lógica funcional** que conecta la interfaz con la base de datos. Implementa la validación de datos, carga de la lista y acciones CRUD.

  Funciones clave:

  - LlenarDatos() → Carga todos los registros desde SQLite al ListView.
  - ValidarDatos() → Comprueba que los campos no estén vacíos.
  - btnRegistrar_Clicked() → Inserta un nuevo profesor.
  - lstProfesores_ItemSelected() → Carga los datos del profesor seleccionado.
  - btnActualizar_Clicked() → Guarda cambios sobre el profesor seleccionado.
  - btnEliminar_Clicked() → Solicita confirmación y elimina el registro.
  - LimpiarCampos() → Restablece el formulario y oculta los botones de edición.

DBSqliteChecklist ▸ DBSqliteChecklist.MainPage ▸ lstProfesores_ItemSelected(object sender, SelectedItemChangedEventArgs e)

```
using Xamarin.Forms;
using DBSqliteChecklist.Models;
using System;

namespace DBSqliteChecklist
{
    5 referencias
    public partial class MainPage : ContentPage
    {
        1 referencia
        public MainPage()
        {
            InitializeComponent();
            LlenarDatos();
        }

        // Registrar
        0 referencias
        private async void btnRegistrar_Clicked(object sender, EventArgs e)
        {
            if (ValidarDatos())
            {
                Profesor prof = new Profesor
                {
                    IdProfesor = Guid.NewGuid().ToString(),
                    Nombre = txtNombre.Text,
                    Apellido = txtApellido.Text
                };

                await DBSqlite.App.SQLiteDB.GuardarProfesorAsync(prof);
                await DisplayAlert("Éxito", "Profesor registrado correctamente", "OK");

                LimpiarCampos();
                LlenarDatos();
            }
            else
                await DisplayAlert("Error", "Por favor complete todos los campos", "OK");
        }

        // Validar
        2 referencias
        bool ValidarDatos()
        {
            return !string.IsNullOrEmpty(txtNombre.Text) &&
                   !string.IsNullOrEmpty(txtApellido.Text);
        }

        // Llenar lista
        4 referencias
```

```
        // Llenar lista
        4 referencias
        private async void LlenarDatos()
        {
            var profesores = await DBSqlite.App.SQLiteDB.ObtenerProfesoresAsync();
            lstProfesores.ItemsSource = profesores;
        }

        // Seleccionar profesor
        0 referencias
        private void lstProfesores_ItemSelected(object sender, SelectedItemChangedEventArgs e)
        {
            if (e.SelectedItem != null)
            {
                var obj = (Profesor)e.SelectedItem;

                txtIdProfesor.Text = obj.IdProfesor;
                txtNombre.Text = obj.Nombre;
                txtApellido.Text = obj.Apellido;

                // Mostrar botones de edición y eliminación
                btnActualizar.IsVisible = true;
                btnEliminar.IsVisible = true;
            }
        }

        // Actualizar
        0 referencias
        private async void btnActualizar_Clicked(object sender, EventArgs e)
        {
            if (ValidarDatos())
            {
                Profesor prof = new Profesor
                {
                    IdProfesor = txtIdProfesor.Text,
                    Nombre = txtNombre.Text,
                    Apellido = txtApellido.Text
                };

                await DBSqlite.App.SQLiteDB.ActualizarProfesorAsync(prof);
                await DisplayAlert("Éxito", "Profesor actualizado correctamente", "OK");

                LimpiarCampos();
                LlenarDatos();
            }
            else
                await DisplayAlert("Error", "Por favor complete todos los campos", "OK");
        }
```

- **App.xaml**

  Define la **estructura global de la aplicación** y los recursos compartidos. En este caso, no contiene estilos ni recursos personalizados, pero es necesario para el funcionamiento de Xamarin.Forms. Solo declara la clase principal DBSqlite.App.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DBSqlite.App">
    <Application.Resources>
        <ResourceDictionary>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

- **App.xaml.cs:**

  Contiene la **lógica de inicialización de la aplicación**, incluyendo la creación de la base de datos local y la asignación de la página principal (MainPage). Aquí se crea una instancia global del helper SQLite que se puede usar en cualquier parte del proyecto.

```csharp
using System;
using System.IO;
using Xamarin.Forms;
using DBSqliteChecklist.Data;

namespace DBSqlite
{
    public partial class App : Application
    {
        static SQLiteHelper db;

        public App()
        {
            InitializeComponent();
            MainPage = new DBSqliteChecklist.MainPage();
        }

        public static SQLiteHelper SQLiteDB
        {
            get
            {
                if (db == null)
                {
                    string dbPath = Path.Combine(
                        Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                        "ChecklistDB.db3"
                    );
                    db = new SQLiteHelper(dbPath);
                }
                return db;
            }
        }
    }
}
```

2. Principales métodos implementados:

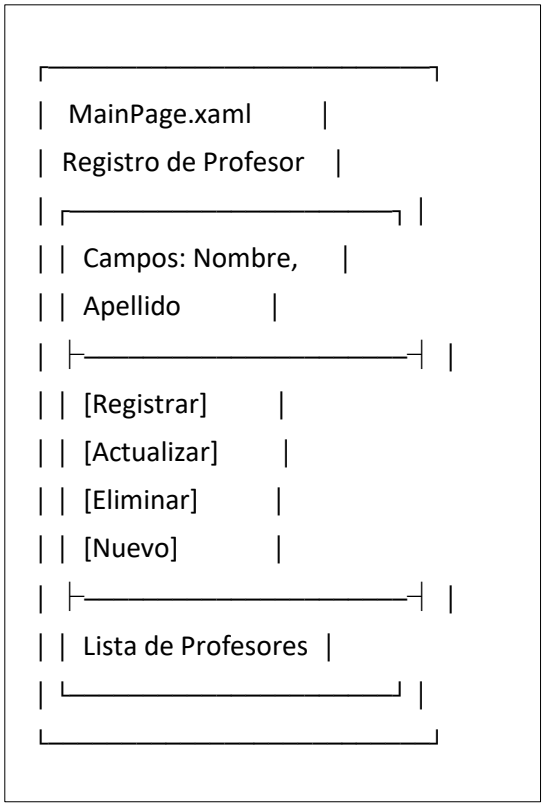
- **SQLiteHelper.cs:**

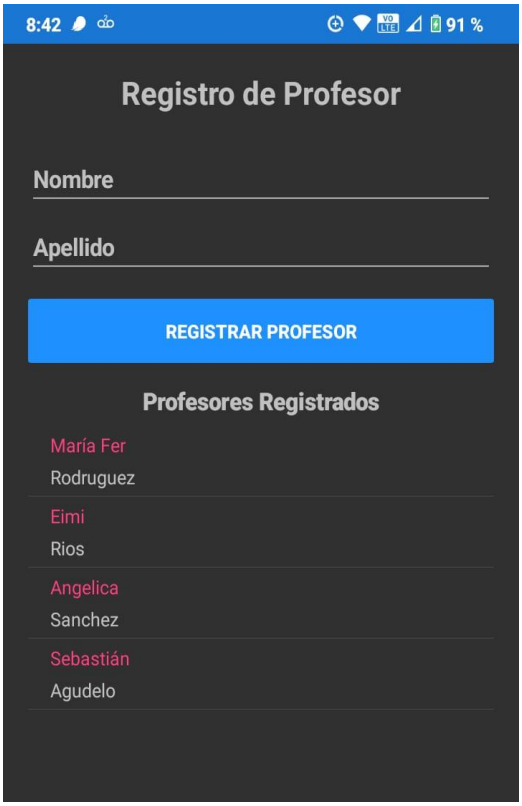
```
public Task<int> GuardarProfesorAsync(Profesor prof)   // Inserta o reemplaza
public Task<List<Profesor>> ObtenerProfesoresAsync()   // Consulta todos
public Task<int> ActualizarProfesorAsync(Profesor prof)// Actualiza
public Task<int> EliminarProfesorAsync(Profesor prof)  // Elimina
```

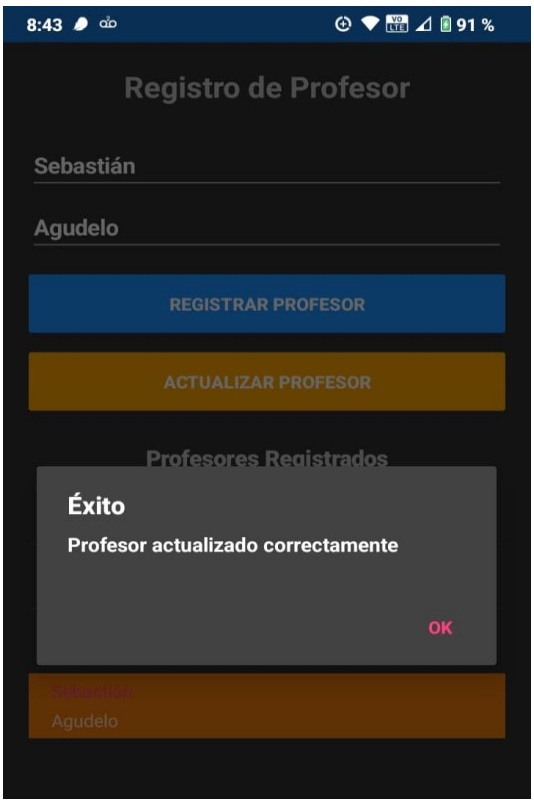
- **MainPage.xaml.cs:**

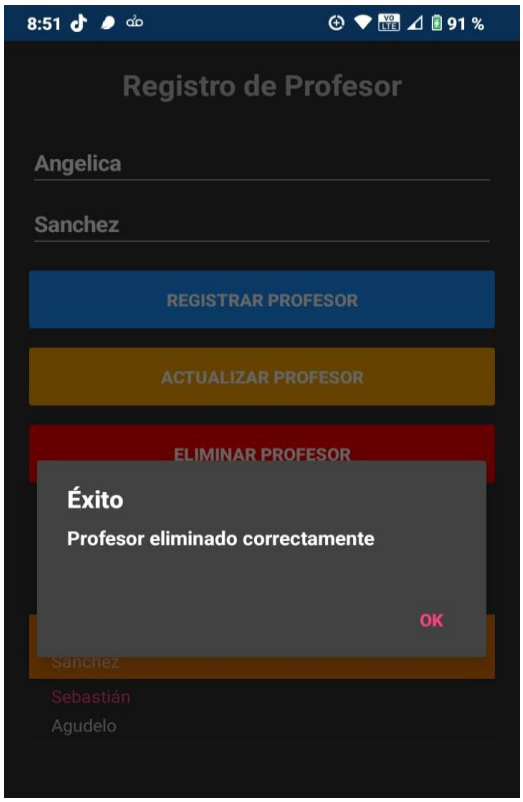
```
private async void LlenarDatos()        // Carga la lista desde SQLite
private bool ValidarDatos()             // Valida campos vacíos
private async void btnRegistrar_Clicked() // Inserta profesor
private async void btnActualizar_Clicked()// Actualiza profesor
private async void btnEliminar_Clicked()  // Elimina profesor
private void LimpiarCampos()            // Limpia formulario
```

3. Diagrama de navegación:









4. Decisiones técnicas:

- **Framework: Xamarin.Forms (Visual Studio 2022):**
  - Permite desarrollar una sola aplicación móvil con C# y XAML que funciona tanto en Android como iOS.
  - Se eligió por su integración directa con Visual Studio y facilidad para trabajar con bases de datos locales.

- **Base de datos: SQLite local (sqlite-net-pcl):**
  - Librería ligera de base de datos embebida, ideal para almacenamiento local sin conexión a internet.
  - No requiere servidor externo, todo se guarda en un archivo .db3 dentro del dispositivo.
  - Compatible con Xamarin.Forms y fácil de implementar mediante clases asíncronas (SQLiteAsyncConnection).

- **Diseño de UI: StackLayout, Entry, Button, ListView:**
  - Se utiliza para definir la interfaz visual: StackLayout, Entry, Button, Label, ListView.
  - Los botones cambian dinámicamente su visibilidad según la acción (por ejemplo, solo mostrar “Actualizar” y “Eliminar” al seleccionar un profesor).

- **Lenguaje: C#**
  - Lenguaje orientado a objetos, estable, seguro y con excelente soporte para manejo asíncrono (async/await).
  - Permite escribir toda la lógica del CRUD y conectarse con la base de datos SQLite fácilmente.

- **Patrón: MVVM simple (Model + Code-behind):**
  - Aunque no se implementa un ViewModel completo, se sigue el principio de separación:
  - Model: Clase Profesor con los atributos.
  - Data: Clase SQLiteHelper para operaciones CRUD.
  - View: MainPage.xaml (interfaz gráfica).
  - Code-behind: MainPage.xaml.cs con la lógica de interacción.

- **Persistencia: Archivo .db3 generado en la carpeta local de la app**
  - Los datos se guardan en un archivo: ChecklistDB.db3 que se crea en la carpeta local de la app:Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "ChecklistDB.db3");

- **Logica de programación:**
  - Uso de métodos asíncronos para no bloquear la interfaz:

```
await db.InsertAsync(profesor);
await db.UpdateAsync(profesor);
await db.DeleteAsync(profesor);
```

  - Uso de validaciones simples antes de cada acción (verifica campos vacíos).
  - Actualización automática del ListView al agregar, editar o eliminar.

- **Entorno de desarrollo:**
  - Visual Studio 2022
  - Proyecto tipo Xamarin.Forms App (.NET Framework).
  - Probado en emulador Android 11.0 y dispositivo físico Android.
  - Usa el SDK de Android, nuget sqlite-net-pcl y soporte para XAML Hot Reload.

- **Estilo visual:**
  - Colores y estructura intuitiva:
    Azul (DodgerBlue) para registrar.
    Naranja (Orange) para actualizar.
    Rojo (Red) para eliminar.
  - Fuente legible y disposición con StackLayout y Margin para buena presentación visual.

- **Gestión de errores y alertas:**
  - Validación previa a cualquier inserción o actualización:

```
if (!string.IsNullOrEmpty(txtNombre.Text) && !string.IsNullOrEmpty(txtApellido.Text))
```

  - Mensajes de usuario con DisplayAlert:
  - “Profesor registrado correctamente”
  - “Por favor complete todos los campos”
  - “¿Deseas eliminar este profesor?”

5. Diseño y experiencia de Usuario:

- Interfaz limpia con colores simples (DodgerBlue, Orange, Red).
- Botones visibles dinámicamente según acción:
- Solo se muestran **Actualizar** y **Eliminar** cuando hay un elemento seleccionado.
- ListView actualiza en tiempo real.
- Mensajes de confirmación (DisplayAlert) para operaciones CRUD.

6. Funcionalidad CRUD completa:

Acción	Descripción	Método principal
Crear	Inserta un nuevo profesor	GuardarProfesorAsync()
Listar	Carga todos los registros	ObtenerProfesoresAsync()
Actualizar	Modifica un registro existente	ActualizarProfesorAsync()
Eliminar	Borra un registro existente	EliminarProfesorAsync()

Conclusión

La aplicación *Checklist* fue desarrollada utilizando **Xamarin.Forms** como framework multiplataforma y **SQLite** como sistema de gestión de base de datos local. Se implementaron las operaciones CRUD (crear, leer, actualizar y eliminar) mediante la librería **sqlite-net-pcl**, con métodos asíncronos para optimizar el rendimiento y evitar bloqueos en la interfaz de usuario.

El proyecto emplea una estructura modular compuesta por las capas **Model**, **Data** y **View**, garantizando una separación lógica entre la definición de datos, la manipulación de la base y la presentación visual. La persistencia se realiza a través de un archivo local .db3, generado dinámicamente en la carpeta interna de la aplicación, lo que permite mantener la información almacenada sin conexión a internet.

El uso de **C#**, **XAML** y **Visual Studio 2022** permitió la integración eficiente de la interfaz gráfica con la lógica funcional, logrando una aplicación estable, escalable y fácilmente mantenible para entornos móviles Android.