SQLite en .NET Standard

1. Agregue los nugets: **SQLite.Net-PCL** y **SQLiteNet Extensions** (versión 1.3), en todos los proyectos del Front.
2. Edita el proyecto compartido y asegúrate que quede similar al ejemplo:

<PropertyGroup>

<TargetFramework>netstandard2.0</TargetFramework>

<PackageTargetFallback>$(PackageTargetFallback);portable-win+net45+wp8+win81+wpa8</PackageTargetFallback>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="MvvmLightLibs" Version="5.3.0" />

<PackageReference Include="Xam.Plugin.Connectivity" Version="3.1.1" />

<PackageReference Include="Xam.Plugins.Settings" Version="3.1.1" />

<PackageReference Include="Xamarin.FFImageLoading" Version="2.3.4" />

<PackageReference Include="Xamarin.FFImageLoading.Forms" Version="2.3.4" />

<PackageReference Include="Xamarin.FFImageLoading.Svg" Version="2.3.4" />

<PackageReference Include="Xamarin.FFImageLoading.Svg.Forms" Version="2.3.4" />

<PackageReference Include="Xamarin.FFImageLoading.Transformations" Version="2.3.4" />

<PackageReference Include="Xamarin.Forms" Version="2.5.0.122203" />

<PackageReference Include="Newtonsoft.Json" Version="11.0.1" />

<PackageReference Include="Xam.Plugins.Forms.ImageCircle" Version="2.0.2" />

<PackageReference Include="Xam.Plugin.Media" Version="3.1.3" />

<PackageReference Include="SQLite.Net-PCL" Version="3.1.1" />

<PackageReference Include="SQLiteNetExtensions" Version="1.3.0" />

</ItemGroup>

1. Crea la interfaz **IConfig**:

namespace Lands.Interfaces

{

using SQLite.Net.Interop;

public interface IConfig

{

string DirectoryDB { get; }

ISQLitePlatform Platform { get; }

}

}

1. Haz la implementación en **Android**:

[assembly: Xamarin.Forms.Dependency(typeof(Lands.Droid.Implementations.Config))]

namespace Lands.Droid.Implementations

{

using Interfaces;

using SQLite.Net.Interop;

public class Config : IConfig

{

private string directoryDB;

private ISQLitePlatform platform;

public string DirectoryDB

{

get

{

if (string.IsNullOrEmpty(directoryDB))

{

directoryDB = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);

}

return directoryDB;

}

}

public ISQLitePlatform Platform

{

get

{

if (platform == null)

{

platform = new SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();

}

return platform;

}

}

}

}

1. Haz la implementación en **iOS**:

[assembly: Xamarin.Forms.Dependency(typeof(Lands.iOS.Implementations.Config))]

namespace Lands.iOS.Implementations

{

using System;

using Interfaces;

using SQLite.Net.Interop;

public class Config : IConfig

{

private string directoryDB;

private ISQLitePlatform platform;

public string DirectoryDB

{

get

{

if (string.IsNullOrEmpty(directoryDB))

{

var directory = Environment.GetFolderPath(Environment.SpecialFolder.Personal);

directoryDB = System.IO.Path.Combine(directory, "..", "Library");

}

return directoryDB;

}

}

public ISQLitePlatform Platform

{

get

{

if (platform == null)

{

platform = new SQLite.Net.Platform.XamarinIOS.SQLitePlatformIOS();

}

return platform;

}

}

}

}

1. Crea el modelo **UserLocal**:

namespace Lands.Models

{

using SQLite.Net.Attributes;

public class UserLocal

{

[PrimaryKey]

public int UserId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public string Telephone { get; set; }

public string ImagePath { get; set; }

public int? UserTypeId { get; set; }

public string ImageFullPath

{

get

{

if (string.IsNullOrEmpty(ImagePath))

{

return "noimage";

}

return string.Format(

"http://landsapi1.azurewebsites.net/{0}",

ImagePath.Substring(1));

}

}

public string FullName

{

get

{

return string.Format("{0} {1}", this.FirstName, this.LastName);

}

}

public override int GetHashCode()

{

return UserId;

}

}

}

1. Crea la clase **DataAccess** en **Helpers**:

namespace Lands.Helpers

{

using Interfaces;

using Models;

using SQLite.Net;

using SQLiteNetExtensions.Extensions;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using Xamarin.Forms;

public class DataAccess : IDisposable

{

private SQLiteConnection connection;

public DataAccess()

{

var config = DependencyService.Get<IConfig>();

this.connection = new SQLiteConnection(

config.Platform,

Path.Combine(config.DirectoryDB, "Lands.db3"));

connection.CreateTable<UserLocal>();

}

public void Insert<T>(T model)

{

this.connection.Insert(model);

}

public void Update<T>(T model)

{

this.connection.Update(model);

}

public void Delete<T>(T model)

{

this.connection.Delete(model);

}

public T First<T>(bool WithChildren) where T : class

{

if (WithChildren)

{

return connection.GetAllWithChildren<T>().FirstOrDefault();

}

else

{

return connection.Table<T>().FirstOrDefault();

}

}

public List<T> GetList<T>(bool WithChildren) where T : class

{

if (WithChildren)

{

return connection.GetAllWithChildren<T>().ToList();

}

else

{

return connection.Table<T>().ToList();

}

}

public T Find<T>(int pk, bool WithChildren) where T : class

{

if (WithChildren)

{

return connection.GetAllWithChildren<T>().FirstOrDefault(m => m.GetHashCode() == pk);

}

else

{

return connection.Table<T>().FirstOrDefault(m => m.GetHashCode() == pk);

}

}

public void Dispose()

{

connection.Dispose();

}

}

}

1. Crea el servicio **DataService**:

namespace Lands.Services

{

using System;

using System.Collections.Generic;

using System.Linq;

using Helpers;

public class DataService

{

public bool DeleteAll<T>() where T : class

{

try

{

using (var da = new DataAccess())

{

var oldRecords = da.GetList<T>(false);

foreach (var oldRecord in oldRecords)

{

da.Delete(oldRecord);

}

}

return true;

}

catch (Exception ex)

{

ex.ToString();

return false;

}

}

public T DeleteAllAndInsert<T>(T model) where T : class

{

try

{

using (var da = new DataAccess())

{

var oldRecords = da.GetList<T>(false);

foreach (var oldRecord in oldRecords)

{

da.Delete(oldRecord);

}

da.Insert(model);

return model;

}

}

catch (Exception ex)

{

ex.ToString();

return model;

}

}

public T InsertOrUpdate<T>(T model) where T : class

{

try

{

using (var da = new DataAccess())

{

var oldRecord = da.Find<T>(model.GetHashCode(), false);

if (oldRecord != null)

{

da.Update(model);

}

else

{

da.Insert(model);

}

return model;

}

}

catch (Exception ex)

{

ex.ToString();

return model;

}

}

public T Insert<T>(T model)

{

using (var da = new DataAccess())

{

da.Insert(model);

return model;

}

}

public T Find<T>(int pk, bool withChildren) where T : class

{

using (var da = new DataAccess())

{

return da.Find<T>(pk, withChildren);

}

}

public T First<T>(bool withChildren) where T : class

{

using (var da = new DataAccess())

{

return da.GetList<T>(withChildren).FirstOrDefault();

}

}

public List<T> Get<T>(bool withChildren) where T : class

{

using (var da = new DataAccess())

{

return da.GetList<T>(withChildren).ToList();

}

}

public void Update<T>(T model)

{

using (var da = new DataAccess())

{

da.Update(model);

}

}

public void Delete<T>(T model)

{

using (var da = new DataAccess())

{

da.Delete(model);

}

}

public void Save<T>(List<T> list) where T : class

{

using (var da = new DataAccess())

{

foreach (var record in list)

{

InsertOrUpdate(record);

}

}

}

}

}

1. Ya estamos listos para grabar datos en una BD SQLite.