Table of Contents

[5.1 SQLite Xamarin Forms 1](#_Toc525135403)

[5.1.1 Entity Framework Core. 2](#_Toc525135404)

[5.2 Ejemplo 2](#_Toc525135405)

[5.2.1 Insertar Nuevo registro 4](#_Toc525135406)

[5.2.2 Actualizar Registro 5](#_Toc525135407)

[5.2.3 Obtener 5](#_Toc525135408)

[5.2.4 Eliminar registro 5](#_Toc525135409)

# 

# 

# 5.1 SQLite Xamarin Forms

Cuando desarrollamos aplicaciones móviles, en una gran cantidad de ocasiones, vamos a necesitar trabajar y gestionar datos. Podemos necesitar sencillas clave-valor para gestionar la configuración de la aplicación, o bien, datos más complejos relacionales utilizando una base de datos local como SQLite.

## 5.1.1 Entity Framework Core.

Entity Framework es un ORM (object-relational mapper) que permite a los desarrolladores trabajar con una base de datos utilizando objetos .NET. Con un largo recorrido, la primera versión de Entity Framework aparecía en verano de 2008, ha ido evolucionando en base a las necesidades de desarrolladores. Como proyecto Open Source, ha contado con importantes contribuciones.

## 5.2 Ejemplo

Tenemos que añadir paquete NuGet de Entity Framework en nuestros proyectos. Para ello, hacemos clic derecho en la gestión de paquetes NuGet de la solución y añadimos Microsoft.EntityFrameworkCore.Sqlite.



Tras añadir el paquete de Entity Framework podemos comenzar a trabajar!. Si ya has trabajado previamente con Entity Framework no te será desconocido el contexto DbContext.

En Entity Framework llamamos contexto a la puerta de enlace entre el modelo que utilizamos y el framework que se encarga de conectar con al base de datos y de realizar el mapeo de las diferentes operaciones con comando de datos. DbContext será el responsable de:

* Conexiones de base de datos.
* Operaciones de datos tales como consultas y persistencia.
* Seguimiento de cambios.
* Mapeo de datos.
* Gestión de transacciones.

Comenzamos definiendo nuestros Modelos que se encargarán de definir el esquema de nuestra base de datos.

1. **public** **class** TodoItem
2. {
3. [Key]
4. **public** **int** Id { **get**; **set**; }
5. **public** **string** Name { **get**; **set**; }
6. **public** **string** Notes { **get**; **set**; }
7. **public** **bool** Done { **get**; **set**; }
8. }

A continuación, nos aseguramos que nuestra clase *TodoItem* es parte de nuestro *DbContext*. Definimos el **contexto**:

1. **class** DatabaseContext : DbContext
2. {
3. **public** DbSet<TodoItem> TodoItems { **get**; **set**; }
5. **public** DatabaseContext()
6. {
7. **this**.Database.EnsureCreated();
8. }
10. **protected** **override** **void** OnConfiguring(DbContextOptionsBuilder optionsBuilder)
11. {
12. var dbPath = DependencyService.Get<IDatabaseService>().GetDatabasePath();
13. optionsBuilder.UseSqlite($"Filename={dbPath}");
14. }
15. }

De igual forma a cuando trabajamos directamente con SQLite por ejemplo, necesitamos código específico por plataforma para acceder a la ruta de la base de datos adecuada.

1. /\*--------Android----------\*/
2. [assembly: Dependency(**typeof**(DatabaseService))]
3. **public** **class** DatabaseService : IDatabaseService
4. {
5. **public** **string** GetDatabasePath()
6. {
7. **return** Path.Combine(
8. System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal),
9. AppSettings.DatabaseName);
10. }
11. }
13. /\*--------IOS----------\*/
14. [assembly: Dependency(**typeof**(DatabaseService))]
15. **public** **class** DatabaseService : IDatabaseService
16. {
17. **public** **string** GetDatabasePath()
18. {
19. **return** Path.Combine(Environment.GetFolderPath(
20. Environment.SpecialFolder.MyDocuments),
21. "..",
22. "Library",
23. AppSettings.DatabaseName);
24. }
25. }

NOTA: Es importante utilizar el servicio de dependencia de Xamarin.Forms parar resolver la dependencia específica de la plataforma donde se ejecuta la aplicación.

### 5.2.1 Insertar Nuevo registro

La clase DbSet<TEntity> representa una colección para una entidad dada dentro del modelo y es la vía de entrada a las operaciones de la base de datos. Las clases DbSet<TEntity> se agregan como propiedades al DbContext.

Para insertar nuevos registros a la colección representada por el DbSet, utilizamos el método DbSet.Add. Esto añade los cambios al DbSet en memoria, es necesario utilizar el método **SaveChanges**para actualizar la base de datos.

1. \_context.TodoItems.Add(item);
2. //
3. \_context.SaveChanges();

### 

### 5.2.2 Actualizar Registro

Para actualizar una entidad, se deben realizar los cambios necesarios en el valor de la propiedad o propiedades de la misma y utilizar el método **Update** junto a  **SaveChanges**.

1. \_context.TodoItems.Update(todoItem);

### 5.2.3 Obtener

La forma más común utilizada para obtener varias entidades es utilizar el método ToList:

1. **public** IList<TodoItem> GetAll()
2. {
3. **return** \_context.TodoItems.ToList();
4. }

Si lo que se busca es recuperar una sola instancia de una entidad, se puede usar el método **First** o **Single**, dependiendo de si espera que haya más de una fila que coincida con los criterios.

### 5.2.4 Eliminar registro

para eliminar registros, utilizamos el método Remove disponible en la clase DbSet además de, SaveChanges.

1. **public** **void** Remove(TodoItem item)
2. {
3. \_context.TodoItems.Remove(item);
4. \_context.SaveChanges();
5. }