Scheduler. *Parte 5 – Complejidad media*

Esta es la quinta parte del ejercicio, hasta aquí hemos demostrado que para desarrollar con calidad no se necesita conocer nada referente a la persistencia de los datos y al frontal relativo a la presentación. A partir de ahora, el reto es intentar aplicar esto que has aprendido en los desarrollos de cada día y para ello debes intentar escribir tus métodos usando Single Responsability y hacer Unit Test siempre que puedas.

Este ejercicio es una extensión para aprender a gestionar nuestro modelo de persistencia con un ORM. En este caso vamos a usar Entity Framework Core, pero existen muchos más como NHibernate, Dapper, XPO de Devexpress, MicroLite, PetaPoco, Massive, etc, cada uno con sus características, los ORM actuales están basados por lo general en modelos de entidades POCO (Plain Old CLR Object), que no incorporan ninguna lógica de negocio. Cada entidad (clase en c#) hace referencia a una tabla de la base de datos relacional y es una imagen de su estructura.

Existen dos aproximaciones distintas para generar nuestro modelo de entidades y contexto de base de datos:

- La primera denominada code first, que consiste en escribir nuestro modelo de datos para luego a través de migraciones generar y actualizar nuestra estructura de base de datos, este modelo es adecuado en nuevos desarrollos y en aquellos que no tengan una alta complejidad.

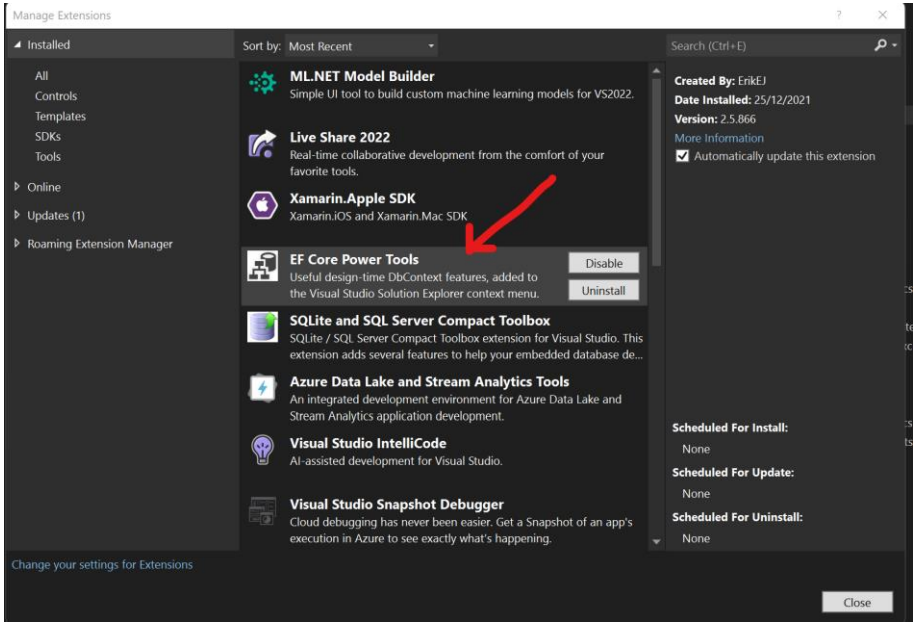
- La segunda usa un modelo de ingeniería inversa para regenerar el modelo a través de la estructura de una base de datos ya creada. Para ello tenemos dos alternativas:

    - La primera usar la línea de comandos para generar un contexto de trabajo. Ejemplo:

      dotnet ef dbcontext scaffold "Data Source=SALAMANDRA\\SQLSERVER2012;Initial Catalog=PRE_PortalInvestigacion;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -d -c "ResearchContext" --context-dir "Context" -o "Model" -n "Semicrol.Fundanet.Model" --use-database-names --no-pluralize.

Un **reto** por aprender **CALIDAD**

- La segunda se basa en usar un el plugin de Erik Ejlskov llamado EF Core Power Tools que os podéis descargar usando las extensiones de visual studio.



El ejercicio consiste en crear una base de datos con **una única tabla** llamada ScheduleConfiguration, que **será capaz de almacenar diferentes configuraciones de schedulers distintos**, para ello creareis un proyecto de clases independiente que almacenará el modelo de datos y sus clases.

La tabla debe tener los datos de la configuración, y debéis añadir dos campos adicionales uno llamado SchedulerId de tipo (autonúmerico) y otro campo adicional de tipo (DateTime) para almacenar la fecha calculada de la próxima ejecución.

Vuestro programa debe ser capaz de **insertar** distintas configuraciones, **actualizarlas y borrarlas a través de un ID** para **devolver una colección con aquellas configuraciones en las que la fecha actual este por encima de la fecha de próxima ejecución**.

Para verificar que vuestro programa funciona como debería deberéis crear test y usar un contexto en memoria InMemory (entityframeworkcore.com) y aplicar un patrón perteneciente a SOLID llamado Inyección de dependencias o pasar el contexto como parámetro como podéis ver en el ejemplo. EF Core testing sample - EF Core | Microsoft Docs

```
[Fact]
public void Can_get_items()
{
    using (var context = new ItemsContext(ContextOptions))
    {
        var controller = new ItemsController(context);

        var items = controller.Get().ToList();

        Assert.Equal(3, items.Count);
        Assert.Equal("ItemOne", items[0].Name);
        Assert.Equal("ItemThree", items[1].Name);
        Assert.Equal("ItemTwo", items[2].Name);
    }
}
```

Podéis comenzar con un pequeño tutorial Introducción - EF Core | Microsoft Docs, pero existen numerosos tutoriales y videos sobre esta tecnología:

Getting Started with Entity Framework Core | Entity Framework Core 101 [1 of 5] - YouTube

Getting started with EF Core in ASP.NET COre - YouTube

Entity Framework Core : Code First | Getting Started - EP01 - YouTube