Integrando un Sistema Gestor de Base de Datos a .NetCore

Requerimientos

- Estar en un sistema *Linux* (en una distribución basada en *Debian*, *Ubuntu* o *Fedora*).
- Tener *.NetCore* instalado en el sistema, en su última version LTS (al momento la elaboración de esta guía es la versión 3.1).
- Tener *Git* instalado en el sistema.
- Un editor de texto o IDE.
- *Postman* o cualquier herramienta para probar APIs.

Instalación del Gestor de Base de Datos SQL

Primero tenemos que instalar un gestor de base de datos SQL:

```
sudo apt install mariadb-server
```

Para este ejemplo se instalará MariaDB

Una vez se instale nuestro Gestor de Base de Datos, ejecutamos las siguientes sentencias *SQL*:

```
DROP USER IF EXISTS gestionarticulos@localhost;
CREATE USER IF NOT EXISTS gestionarticulos@localhost
IDENTIFIED BY 'gestionarticulos';

DROP DATABASE IF EXISTS gestionarticulos;
CREATE DATABASE IF NOT EXISTS gestionarticulos
DEFAULT CHARACTER SET utf8;

GRANT ALL PRIVILEGES ON gestionarticulos.*
TO gestionarticulos@localhost;
FLUSH PRIVILEGES;
```

Para simpleza de esta guía ocuparemos el mismo nombre para usuario, nombre de la base de datos y contraseña.

Instalación de EntityFrameworkCore

En nuestro proyecto ejecutamos los siguientes comandos:

```
dotnet tool install --global dotnet-ef
dotnet add package Microsoft.EntityFrameworkCore.Design
```

```
dotnet add package Pomelo.EntityFrameworkCore.MySql --version 3.1.1
dotnet add package Microsoft.EntityFrameworkCore.Proxies --version 3.1.3
```

NOTA: El comando *dotnet tool install –global dotnet-ef* ya no se tendrá que ejecutar la proxima vez que se cree un nuevo proyecto.

ConnectionStrings

En nuestro *appsettings.Development.json* hay que introducir las siguientes líneas:

```
"ConnectionStrings": {
    "DefaultConnection": "server=localhost;database=gestionarticulos;
    uid=gestionarticulos;password=gestionarticulos"
    // Importante: Esto va en la misma linea
},
```

NOTA: Esta *ConnectionString* solo servirá para el el perfil de desarrollo (el que se ejecuta con: *dotnet run*), para el despliegue el *ConnectionString* debe ir en el archivo *appsettings.json*.

Creando un DBContext

En nuestra carpeta de *Models*, crearemos la clase *GestionArticulosContext*, esta debería verse algo así:

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace MiPrimeraApi.Models
{
    public class GestionArticulosContext : DbContext
    {
        public GestionArticulosContext
        (DbContextOptions<GestionArticulosContext> opciones)
        : base(opciones) { }
        // IMPORTANTE: Esto va en la misma linea

        // Este es el mapeo objeto-base de datos
        public DbSet<Articulo> Articulos { set; get; }
    }
}
```

Obteniendo la conexión

Agregamos las siguientes dependencias:

```
using Microsoft.EntityFrameworkCore;
```

En nuestro archivo *Startup.cs*, en el método *ConfigureServices* agregamos las siguientes líneas al inicio del método:

```
// Esta línea específica que creará
// tantas instancias de *GestionArticulosContext*
// como sean necesarias.
services.AddDbContextPool<GestionArticulosContext>(options =>
                    options.UseLazyLoadingProxies()
                        .UseMySql(Configuration.
                            GetConnectionString("DefaultConnection")));
```

Haciendo la primera migración

Las migraciones pueden ser vistas como algo similar a un *commit*, donde todos los cambios que se efectuen en *GestionarArticulosContext*, así como cambios en las clases que se agreguen a un *DbSet* se verán reflejadas en la migración.

Para hacer una migración se ocupa el siguiente comando:

```
dotnet ef migrations add CreacionArticulo
```

Este comando creará una carpeta Migrations con varias clase, si accedemos al archivo *###_CreacionArticulo.cs*, se puede observar que se define la creación de la tabla* Articulos*.

Una vez que hemos hecho nuestra migración, sólo queda sincronizar con la base de datos, para eso se ejecuta el siguiente comando:

```
dotnet ef database update
```

Una vez ejecutado ese comando, si se desea se puede observar en el gestor de base de datos que se creo esta tabla.

> NOTA: Al igual que con los commits, las migraciones deben de ser pequenas para facilitar su control.

Incorporando *EntityFrameworkCore* a nuestro controlador

En nuestro controlador de Articulo agregamos el siguiente atributo de clase:

```
private readonly SisManContext _contexto;
```

El constructor se modificaría de la siguiente manera:

```
// Con esto estamos declarando que se aplique
// inyeccion de dependencias de GestionArticulosContext
public ArticuloController(GestionArticulosContext contexto)
{
    _contexto = contexto;
}
```

Ahora nuestro método para obtener todos quedaría algo asi:

```
[HttpGet]
[Route("")]
public IActionResult Obtener()
{
    var articulos = _contexto.Articulos.ToList();
    return Ok(articulos);
}
```

Si se quiere, se puede ejecutar la aplicacion, aunque este método retornará una lista vacia.

Para registrar un nuevo artículo quedaría de la siguiente manera:

```
[HttpPost]
[Route("")]
public IActionResult Registrar(Articulo articulo)
{
    // Hay que recordar que a la hora de hacer un post
    // no hace falta registrar el ID, pues este es generado automáticamente.
    articulo.FechaRegistro = DateTime.Now;
    _contexto.Articulos.Add(articulo);
    _contexto.SaveChanges();
    return CreatedAtAction(nameof(ObtenerPorId), new {articulo.Id}, articulo);
}
```

El/la estudiante deberá implementar el resto de los métodos, asi como los de Proveedor