laboratorio

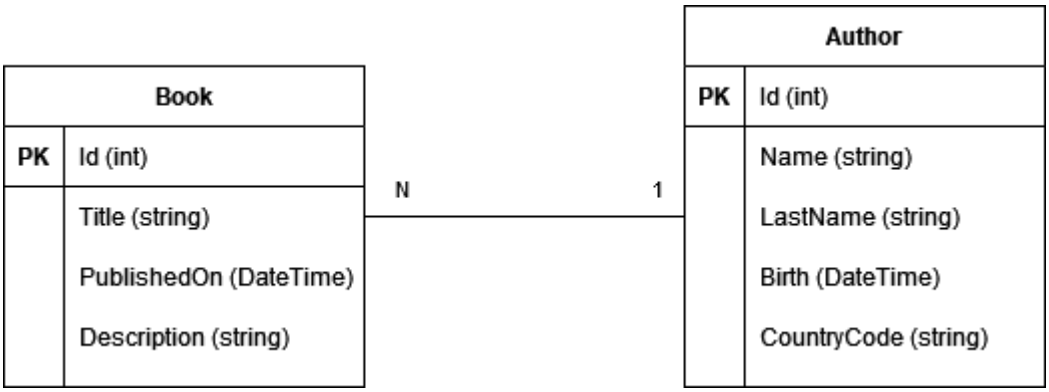
Laboratorio de API REST con persistencia en base de datos relacional SQL Server. Sigue las instrucciones y sube tu código a un repositorio git de tu elección.

Requisitos

- SDK de .NET 7
- EF CLI
- Docker
- SQL Server (como contenedor Docker)

Instrucciones

El ejercicio consiste en crear una API REST para la gestión de libros, con persistencia en SQL Server utilizando Entity Framework. Se parte del siguiente modelo Entidad Relación



Completa cada uno de los siguientes puntos teniendo en cuenta que es obligatorio completar del 1 al 8, es opcional completar del 9 al 11 y la consecución del 12 al 14 sería un reto opcional más avanzado.

1. Descarga los archivos de este repositorio como un zip y descomprímelos en tu propio repositorio git donde subirás cambios para su evaluación.
2. Ejecuta `create-structure.bat` y verifica que se crea una solución con proyectos básicos. Abre la solución. Elimina el soporte para https y de IIS.
3. Haz un `git add .`, `git commit -m "solución inicial"` y `git push` a tu repositorio git remoto en GitLab o GitHub y verifica que los cambios se suben correctamente.
4. Crea todo lo necesario para una creación de migración de base de datos en SQL Server con *code first* y la EF CLI. Sube los cambios a tu repositorio git.
5. Crea un endpoint para añadir Autores (i.e: `POST api/authors`) que acepte payload en formato json para crear nuevos autores con nombre, apellido, fecha de nacimiento y nacionalidad como country code de dos caracteres *ISO 3166-1 alpha-2 code*. Sube los cambios a tu repositorio git.
6. Crea un endpoint para añadir Libros (i.e: `POST api/books`) que acepte payload en formato json para crear nuevos libros con título, fecha de publicación y descripción. Además debe aceptar el identificador del autor y asociarse con ese autor en concreto. Sube los cambios a tu repositorio git.
7. Crea un endpoint para modificar el título o la descripción de un libro (i.e: `PUT api/books/{bookId}`) dado su identificador único. Sube los cambios a tu repositorio git.

8. Crea un endpoint para consultar todos los libros de la base de datos (i.e: `GET api/books`) en el que se devuelve una respuesta en formato json que contiene una colección de objetos con la siguiente información, y sube los cambios a tu repositorio git.
    - Id: el identificador único del libro
    - Title: el título del libro
    - Description: la descripción del libro
    - PublishedOn: la fecha en formato ISO_8601 UTC
    - Author: el nombre completo
9. Añade Swagger/OpenAPI (i.e: paquete nuget `Swashbuckle.AspNetCore`) para interactuar con los endpoints desde la url `/swagger`. Sube los cambios a tu repositorio git.
10. Añade un test funcional donde se valide que: Dada la creación por http de un autor, Cuando se lee de la base de datos el autor con el Id generado, Entonces devuelve el autor que se había creado. Sube los cambios a tu repositorio git.
11. Añade un test funcional donde se valide que: Dada la creación de un autor que tiene dos libros, Cuando se lee a través del endpoint de http la lista de libros, Entonces debería haber dos, con los datos esperados tanto del autor como de los libros. Sube los cambios a tu repositorio git.
12. Añade filtro como query parameter en `GET api/books?title=foo` que busque libros que contengan solamente lo indicado en el título. Sube los cambios a tu repositorio git.
13. Añade otro filtro como query parameter en `GET api/books?title=foo&author=bar` que busque libros que contengan solamente lo indicado en el título y además contenga el nombre o el apellido del autor indicado. Sube los cambios a tu repositorio git.
14. Añade autenticación básica con un filtro o middleware, con credenciales configurables en el `appsettings.json` para proteger todos los endpoint y permitir que solamente las peticiones http que contengan una cabecera `Authorization: Basic <base_64_credentials>` puedan acceder, y modifica Swagger y los tests para que todo pueda seguir funcionando si se incluye autenticación básica. Sube los cambios a tu repositorio git.

Se valorará:

- la limpieza del código y la separación de responsabilidades
- la elaboración de pruebas automáticas
- la consecución de los objetivos
- una buena descripción de cómo ejecutar y lanzar la aplicación para cualquier persona que quiera descargarse el código

Anexos

A continuación se describen herramientas a utilizar y varias utilidades que permitan alcanzar los objetivos para esta aplicación web.

SQL Server con Docker

Para arrancar un servidor de SQL Server 2019 en local con Docker y credenciales `sa / Lem0nC0de!`, ejecuta lo siguiente:

```
docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=Lem0nCode!" -p 1433:1433 --
name sqlserver2019 -d mcr.microsoft.com/mssql/server:2019-latest
```

Si hay algún error, se pueden consultar las trazas con `docker logs sqlserver2019` para ver cuál es el error que hace que el contenedor no arranque.

NOTA: Si ya tienes un contenedor, creado previamente, llamado `sqlserver2019`, arráncalo con

```
docker start sqlserver2019
```

NOTA: Opcionalmente instala una extensión en VSCODE llamada SQL Server (mssql) para conectar y ver base de datos.

Conexión a SQL Server

En visual Studio Community Edition, ve a *View* > *SQL Server Object Explorer* y crea una nueva conexión a Server Name: `localhost` Authentication: `SQL Server Authentication` User Name: `sa` Password: `Lem0nCode!` Database Name: `<default>`

Connection string

Para conectar código con la base de datos, utiliza el siguiente connection string:

```
"Server=localhost;Database=Books;user=sa;password=Lem0nCode!;Encrypt=False"
```

NOTA: Este connection string asume que el nombre de la base de datos a utilizar es `Chats`

Instalación de la herramienta Dotnet EF

Si ya tienes el SDK de .NET 7 (i.e: `dotnet --version` muestra `7.0.100` o superior), instala la CLI de Entity Framework Core con

```
dotnet tool install --global dotnet-ef
```

y compruébalo con

```
dotnet ef --version
```

que debería mostrar algo como

```
Entity Framework Core .NET Command-line Tools
7.0.0
```

Si tenías una versión anterior, la puedes actualizar con

```
dotnet tool update --global dotnet-ef
```

Migraciones

Al utilizar La inicial se crea con

```
dotnet ef migrations add inicial --project src/<project_containing_db_context> --
startup-project src/<main_project>
```

Un directoro llamado `Migrations` será automáticamente creado. Para aplicar las migraciones generadas se ejecuta:

```
dotnet ef database update --project src/<project_containing_db_context> --startup-
project src/<main_project>
```