Exercice 3

Entity Framework Core : utilisation de EF (ORM)

Objectif : définir un modèle de données Objet dans votre application et construire la base de données correspondante à partir des outils de migration de dotnet. Et travailler sur les données issues de la base de données.

Introduction :

Principe d'un ORM : en Java ce sont JPA et Hibernate qui permettent de garantir cette fonctionnalité d'ORM (Object Relational Mapping) en C# ce sont les EF Entity Framework. Le principe d'un ORM est de faciliter la mise en place de la persistance des données dans un modèle relationnel en utilisant des frameworks qui vont permettre de s'occuper de la transformation de données du modèle Objet de votre application au modèle relationnel de votre base de données en fournissant un ensemble de classes et de méthodes pour effectuer ces opérations. L'objectif est de simplifier le travail du développeur en délégant la partie base de données à ces frameworks ce qui lui permet de se concentrer sur l'activité métier de l'application. Chaque framework a ses spécificités de mise en œuvre et d'utilisation. L'objet de cet exercice est d'utiliser EF Core de .NET dans un exercice tiré de Microsoft et ensuite de voir son utilisation dans notre propre projet.

Pré-requis : votre conteneur SqlServer installé et accessible.

Voir les sites suivants pour comprendre les concepts de base :

https://docs.microsoft.com/fr-fr/ef/#pivot=efcore
https://docs.microsoft.com/en-us/ef/
https://www.supinfo.com/articles/single/6960-introduction-entity-framework-core

Construire un petit projet en mode console. Ensuite veuillez définir dans un ou plusieurs fichiers le modèle de données suivant ou directement dans Program.cs.
Attention aux packages à installer dans le projet

#dotnet add package Microsoft.EntityFrameworkCore.SqlServer
#dotnet add package Microsoft.EntityFrameworkCore.Design

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        optionsBuilder.UseSqlServer(
            @"Server=127.0.0.1;Initial Catalog=epsi;User
ID=sa;Password=RjBnc450");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public int Rating { get; set; }
    public List<Post> Posts { get; set; } = new List<Post>();
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

A configurer en fonction de votre contexte et de vos installations

Regardez le rôle de DbContext dans la partie ORM → permet de définir un contexte de base de données et de garantir l'ouverture d'une session avec la base et de fournir les méthodes nécessaires à la manipulation des données (lecture, modification et suppression). Uniquement grâce à la chaine de connexion et à tous les assemblies (**Microsoft.EntityFrameworkCore.SqlServer et Microsoft.EntityFrameworkCore.Design)** complémentaires en fonction du type de base de données. Ensuite on peut utiliser les DbSet pour faire les chargements directs de la table en collection d'objets et ainsi permettre à partir de l'application de les manipuler.

Ne pas oublier d'ajouter les using nécessaires pour ce type de manipulation.

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

Ensuite à partir de ce modèle on va construire à l'aide des instruction suivantes la base de données permettant de stocker ces informations :
Au préalable il faut installer dotnet ef dans votre environnement à l'aide de la commande #dotnet tool install --global dotnet-ef (voir le contenu de votre répertoire home/.dotnet. Attention à la version de dotnet (3,1).

#dotnet ef migrations add InitialCreate
Création dans le répertoire Migrations des fichiers de traitements

```
├── 20191209201320_InitialCreate.cs
├── 20191209201320_InitialCreate.Designer.cs
└── BloggingContextModelSnapshot.cs
```

#dotnet ef database update

On pourra vérifier directement dans la base la création des tables ...

Ensuite on peut implémenter des requêtes en linq dans le Main par exemple de votre programme …

```
using (var db = new BloggingContext())
{
   var blog = new Blog { Url = "http://automathon.univ-lille.fr",Rating=12 };
   db.Blogs.Add(blog);
   db.SaveChanges();
}
using (var db = new BloggingContext())
{
   var blogs = db.Blogs
      .Where(b => b.Rating > 3)
      .OrderBy(b => b.Url)
      .ToList();

      foreach(Blog b in blogs) {
         Console.WriteLine(b.Url);
      }
}
```

On doit pouvoir implémenter toutes les fonctions de traitement associées. Si on veut après avoir modifier le modèle refaire l'opération il suffit de donner un autre nom à la migration et de refaire l'update.

#dotnet ef migrations add DeuxièmeNom
#dotnet ef database update

Dans le cadre de votre projet établissement, fort de cette expérience, essayez de procéder de la même manière c'est à dire à partir du moèle objets refaire le modèle relationnel.

Quelques aménagements (définition d'une clé primaire dans vos classes) seront nécessaires pour caler avec les contraintes de base de données.
N'oubliez de créer une classe GestionContexte.cs qui héritera de DbContext et qui permettra d'instancier une instance et d'ouvrir une session sur votre base de données.

Déclarer les éléments DbSet pour chaque type d'objet que vous voulez récupérer.

```
public class GestionContext : DbContext
   {
      public DbSet<Etudiant> Etudiants { get; set; }
      public DbSet<Filiere> Filieres { get; set; }
      public DbSet<Enseignant> Enseignants { get; set; }
      protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
      {

         optionsBuilder.UseSqlServer(
            @"Server=127.0.0.1;Initial Catalog=epsi;User
ID=sa;Password=RjBnc450");
      }
   }
```

Adaptation de votre Main :

```
using (var context = new GestionContext()) {
            context.Filieres.Add(fil1);
            context.Filieres.Add(fil2);
            context.Enseignants.Add(ens1);
            context.Enseignants.Add(ens2);
            context.SaveChanges();
         }
         using (var context = new GestionContext()) {
            var Listetu = context.Etudiants
               .ToList();
            foreach(Etudiant etu in Listetu) {
               Console.WriteLine(etu);
            }
         }
```