(Leçon 05)
Introduction à Entity Framework

LOTFI DERBALI, PH.D
DERBALI.IT@GMAIL.COM
HTTPS://GITHUB.COM/DERBALI-IT

Contenu

- Définitions
- Entity Framework (EF)
- Utilisation de EF
- Expression lambda

Définitions

- EF: Qu'est-ce que c'est?
  - Entity Framework permet aux développeurs de créer des applications d'accès aux données en programmant par rapport à un modèle d'application conceptuel au lieu de programmer directement par rapport à un schéma de stockage relationnel.

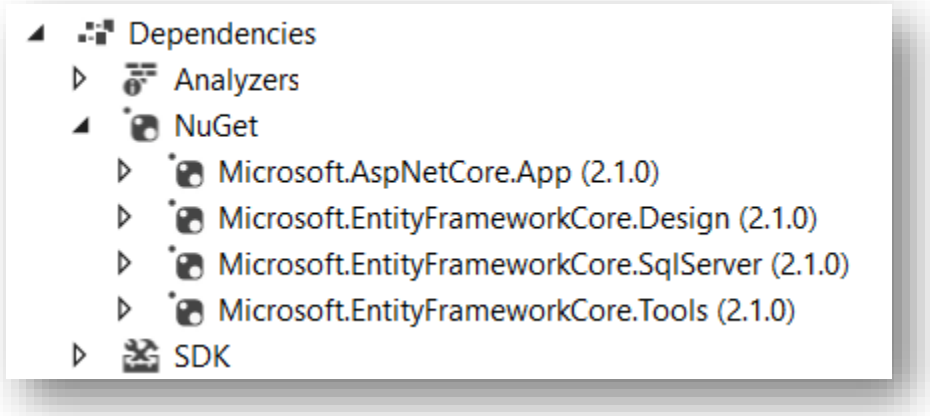
- ORM: Qu'est-ce que c'est?
  - Un mapping objet-relationnel (en anglais Object-Relational Mapping ou ORM) est un type de programme informatique qui se place en interface entre un programme applicatif et une base de données relationnelle pour simuler une base de données orientée objet.
  - Ce programme définit des correspondances entre les schémas de la base de données et les classes du programme applicatif.

Entity Framework

- Entity Framework (EF)
  - EF est le ORM utilisé par la technologie Microsoft .NET
- Avantages (par rapport à ADO.NET)
  - typage
  - manipulation d'objets métier (plutôt que SQL)
- Configurations possibles pour EF
  - Database First
  - Model First
  - Code First

- Database First
  - Vous générez tout le modèle .NET à partir de la base de données
  - Cela implique d'avoir déjà une base de données avec des tables et des relations.

- Étapes:
  1. Créer un projet ASP.NET Core MVC
  2. Installer Entity Framework
     - Dans Visual Studio, faites un clic-droit sur la racine du projet
     - Sélectionner « Gérer les packages NuGet »
     - Installer les 3 packages suivants:
       - Microsoft.EntityFrameworkCore.SqlServer
       - Microsoft.EntityFrameworkCore.Tools
       - Microsoft.EntityFrameworkCore.Design



- Un mot sur le gestionnaire de package **NuGet**
  - Pouvoir se créer des bibliothèques réutilisables est une grande force du .NET.
  - Un gestionnaire de bibliothèques externes : NuGet.
  - NuGet est gratuit et open source, et est fourni en tant qu'extension de Visual Studio.
  - Il est très facile à utiliser, et très pratique quand on a besoin de manipuler des bibliothèques tierces, et notamment des projets de communauté.

Génération du modèle

- Créons à présent le modèle EF à partir de votre base de données existante.
  - **Outils –> Gestionnaire de package NuGet –> Console du Gestionnaire de package**
  - Exécutez, par exemple, la commande suivante

```
Scaffold-DbContext "Server=.\SQLEXPRESS;Database=db_test;User ID=etudiant;
Password=cgodin2018" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

  - Le processus d'ingénierie EF créé des classes d'entité (par exemple TblStudent.cs) et un contexte dérivé (dbtestContext.cs) en fonction du schéma de la base de données existante.

- Autre commande (Si Scaffold-DbContext ne fonctionne pas)
  - Ouvrir CMD.exe de Windows
  - Accéder à la racie de votre projet
    - cd C:\Users\derbali-it\source\repos\Solutionnaire_net\Atelier_5\Atelier_5
  - Lancer la commande suivante:
    - dotnet ef dbcontext scaffold "Server=.\SQLEXPRESS;Database=db_cgodin;User ID=etudiant; Password=cgodin2018" Microsoft.EntityFrameworkCore.SqlServer - -output-dir Models

- Les classes d'entité sont des objets C# simples qui représentent les données que vous allez interroger et enregistrer.
  - Par exemple

```
public partial class TblStudent
    {
        public TblStudent()
        {
        }

        public int Id { get; set; }
        public string Name { get; set; }
        public decimal? Mark { get; set; }

    }
```

- Le contexte représente une session avec la base de données et vous permet d'interroger et d'enregistrer les instances des classes d'entité.

```
public partial class dbtestContext : DbContext
   {
       public dbtestContext()
       {
       }

       public dbtestContext(DbContextOptions<dbtestContext> options)
           : base(options)
       {
       }

       public virtual DbSet<TblStudent> TblStudent { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
      {
            . . .
        }

      protected override void OnModelCreating(ModelBuilder modelBuilder)
      {
            . . .
        }
```

- Remarques:
  - Dans la classe du contexte, on note au passage l'utilisation du type **DbSet** pour marquer l'existence d'une collections.
  - On trouvera généralement autant de **DbSet** que de tables dans la base de données, un par table.
  - Ajouter la configuration de la chaîne de connexion à la base de données dans le fichier **appsettings.json**

```
"ConnectionStrings": {
  "MyConnection": "Server=.\\SQLEXPRESS;Database=dbtest;User ID=etudiant;Password=cgodin2018"
},
```

Relations entre tables

- EF génère aussi les relations entre les entités.

```
class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
        public virtual List<Post> Posts { get; set; }
    }
```

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

- relations 1-1, 1-n ou n-m
- *virtual* = chargement des données uniquement si accédées

Inscrire le contexte avec l'injection de dépendances

- En ASP.NET Core, des services sont inscrits avec l'injection de dépendances au démarrage de l'application.
  - La configuration s'effectue généralement dans le fichier **Startup.cs**.
- Étapes:
  - Ouvrez le fichier du contexte (dans l'exemple, dbtestContext.cs)
  - Commenter la méthode OnConfiguring(...)
  - Ouvrez **Startup.cs**
  - Localisez la méthode ConfigureServices(...)
  - Ajoutez le code suivant pour inscrire le contexte en tant que service.

```
 // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {

            services.AddDbContext<dbtestContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("MyConnection")));
```

Utilisation de EF

- Exploitation de la base de données avec Entity Framework
  - Ajouter, supprimer, modifier des éléments de la base
  - Interroger la base avec des requêtes LINQ sur les entités

- Pour cela, créer un contrôleur et ajouter les actions voulues
  - Dans l'**Explorateur de solutions**, cliquez avec le bouton de droite sur le dossier **Controllers** et sélectionnez **Ajouter -> Contrôleur...**

- Par exemple, le contrôleur *StudentController*

```
public class StudentController : Controller
    {
        private dbtestContext context;

        public StudentController(dbtestContext context)
        {
            this.context = context;
        }

        public IActionResult Index()
        {
          . . .
```

Affichage du contenu d'une table de la base

Utilisation de la DbSet (Collection) dans le contexte

- Par exemple, pour afficher tous les étudiants dans la table tbl_students

```
public class StudentController : Controller
    {
        private dbtestContext context;

        public StudentController(dbtestContext context)
        {
            this.context = context;
        }

        public IActionResult Index()
        {
            //affichage sous forme d'une table
            string body = "<table border=1>";
            foreach (var student in context.TblStudent)
                {
                    body += "<tr>";
                    body += "<td>" + student.Name + " | " + student.Mark + "</td>";
                    body += "</tr>";
                }
            body += "</table>";

            return Content(body, "text/html");
        }
```

Ajout d'éléments au contexte de persistance

- Utilisation de la méthode Add()
- N'oublier pas de sauvegarder le contexte avec la méthode SaveChanges()
  - Par exemple, pour ajouter un nouvel étudiant dans la table tbl_students, ajouter dans le contrôleur StudentController l'action suivante:

```
public IActionResult AddStudent(string name, decimal mark)
        {
                TblStudent student = new TblStudent();
                student.Name = name;
                student.Mark = mark;

                context.Add(student);
                context.SaveChanges();
                return RedirectToAction("Index");
        }
```

Suppression d'éléments du contexte de persistance

- Utilisation de la méthode Remove()
- N'oublier pas de sauvegarder le contexte avec la méthode SaveChanges()
  - Par exemple, pour vider la table tbl_students, ajouter dans le contrôleur StudentController l'action suivante:

```
public IActionResult RemoveAllStudents()
        {
                // on vide la table actuelle des students
                foreach (var student in context.TblStudent)
                {
                    context.TblStudent.Remove(student);
                }
                // on sauve le contexte de persistance
                context.SaveChanges();

                return RedirectToAction("Index");
        }
```

- Autres méthodes (à pratiquer) du contexte de persistance
  - Update()
  - [DbSet].Find()
  - [DbSet].FirstorDefault()
  - [DbSet].Select()
  - [DbSet].Take()
  - [DbSet].OrderBy()
  - …

    - Exemple, afficher les 5 premiers étudiants da la table

```
context.TblStudent.Take(5)
```

Expression lambda

- Une expression lambda est une fonction anonyme qui peut contenir des expressions et des instructions.
- Elle peut ainsi utilisée pour créer des délégués ou des types d'arborescence d'expression.
- Toutes les expressions utilisent l'opérateur lambda **=>** (se lit « conduit à »).

- Une expression est toujours constituée de deux parties :
  - le côté gauche donne les paramètres d'entrées (s'il y en a),
  - le côté doit donne les instructions de la méthode anonyme.

- Exemple 1:

```
int[] nombres = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
// Calcule le nombre d'éléments pairs dans la liste
int pairs = nombres.Count(n => n % 2 == 0);
```

- Exemple 2:

```
string[] animaux = { "dog", "cat", "perls" };
double moyenne = animaux.Average(x => x.Length);
```

Exemples en EF

- Charger toutes les données d'une entité

```
List<Client> clients = context.Clients.ToList();
```

- Charger une seule donnée d'une entité

```
Client client = context.Clients.Single(c => c.ID==3);
```

- Filtrer les données d'une entité

```
List<Client> clientsDeMexico = context.Clients.Where(c =>
c.Country.Equals("Mexico")).ToList();
```

- Autres méthodes pour interroger la source de données
  - Select, SelectMany
  - Find
  - Where
  - GroupJoin, Join
  - All, Any
  - Distinct
  - Except
  - Intersect
  - Union
  - OrderBy, OrderByDescending, ThenBy, ThenByDescending
  - GroupBy
  - Average
  - Count, LongCount
  - Max, Min, Sum
  - OfType, Cast
  - First, FirstOrDefault
  - Skip, Take

Cette liste est non exhaustive