Entity Framework Core

J. ROMAGNY *(Version actuelle 2.0.1 – 11/12/2017)*

Github, documentation

Liens utiles : learnentityframeworkcore.com

Table des matières

1. **Création de projet** (avec Visual Studio ou CLI)

   - On peut créer une **librairie** « Data » pour structurer le projet : « **.Net Core** » **ou** « **.Net Standard** » (**WPF**, **Windows Form**, …). Ajout des **packages NuGet** (« Microsoft.EntityFrameworkCore.SqlServer » (providers) et « Microsoft.EntityFrameworkCore.Tools » (pour migrations)

   - Pour une application **Uwp** (version « 16299 » minimale) et ajouter le package NuGet « Microsoft.EntityFrameworkCore.Sqlite »

   - On peut créer directement une « **application Web Asp.Net Core** ». Pas besoin de packages NuGet sauf CLI

Pour CLI ajouter « Microsoft.EntityFrameworkCore.Tools.DotNet » et éditer le « csproj »

2. Création et configuration **DBContext**, entités

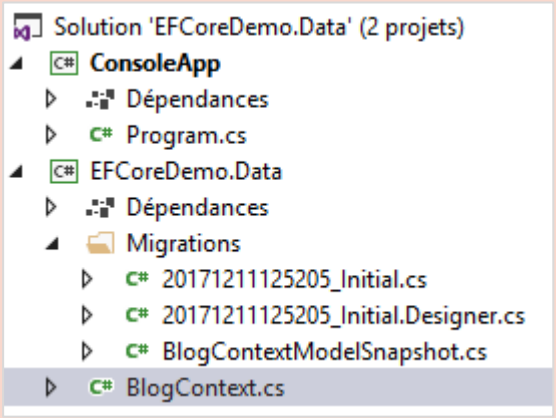
3. Création de la **migration** « **Initial** » et **mise à jour de la base de données**

Si le projet de base est une librairie, ajouter un projet client (exemple une application console), définir ce projet en tant que projet de démarrage et référencer la librairie.

En cas de client « lourd » (WPF par exemple) installer le package «Microsoft.EntityFrameworkCore.SqlServer »

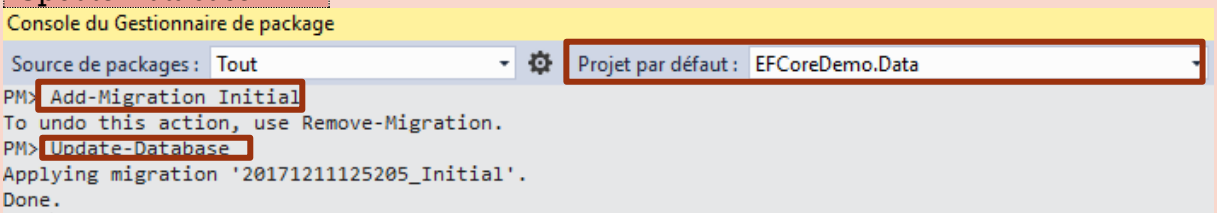
Avec la console du Gestionnaire de package

```
Add-Migration Initial
```



Mise à jour de la base de données

```
Update-Database
```



Ou avec le **CLI** (Pas compatible avec .Net Standard)
Ajouter au projet le package NuGet « Microsoft.EntityFrameworkCore.Tools.DotNet » et éditer le « csproj »

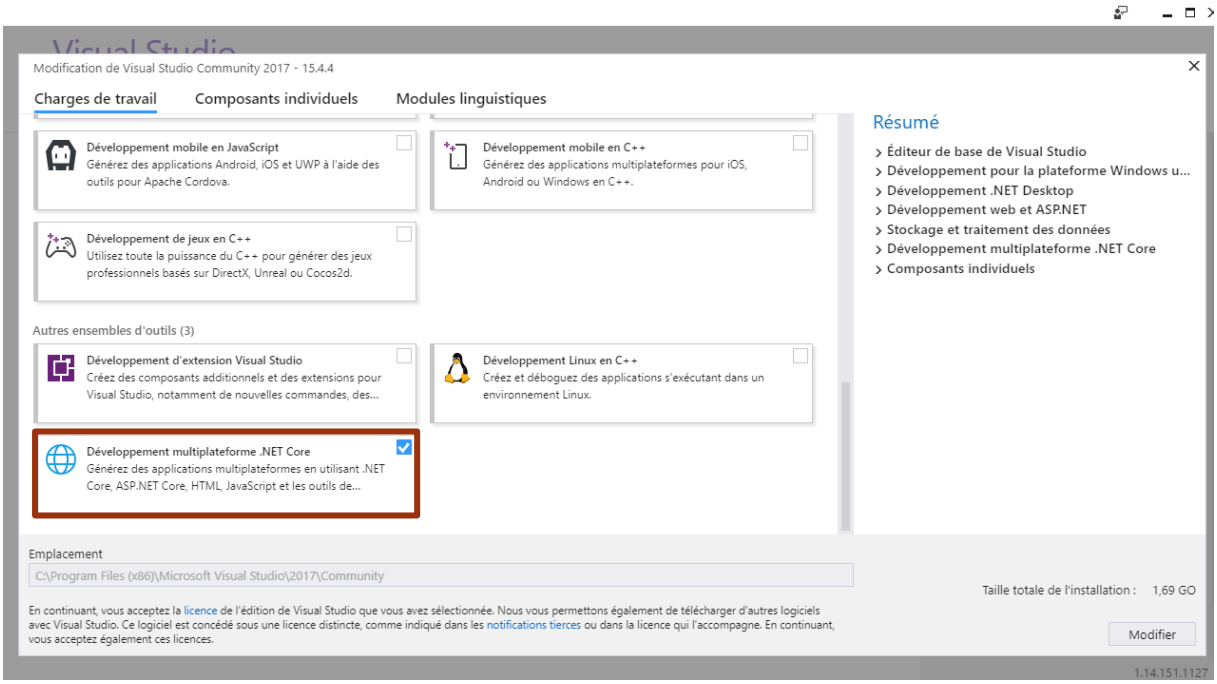
Répéter ensuite « Add-Migration … » avec un nom (exemple « Add-Migration AddUsers ») et « Update-Database »

4. **CRUD avec le dbContext**

1. Installation de .Net Core

2 possibilités :

⇨ Avec **Visual Studio 2017**



(.Net Core dans les « Composants individuels », section « .Net »)

⇨ Ou **téléchargement** du **SDK**

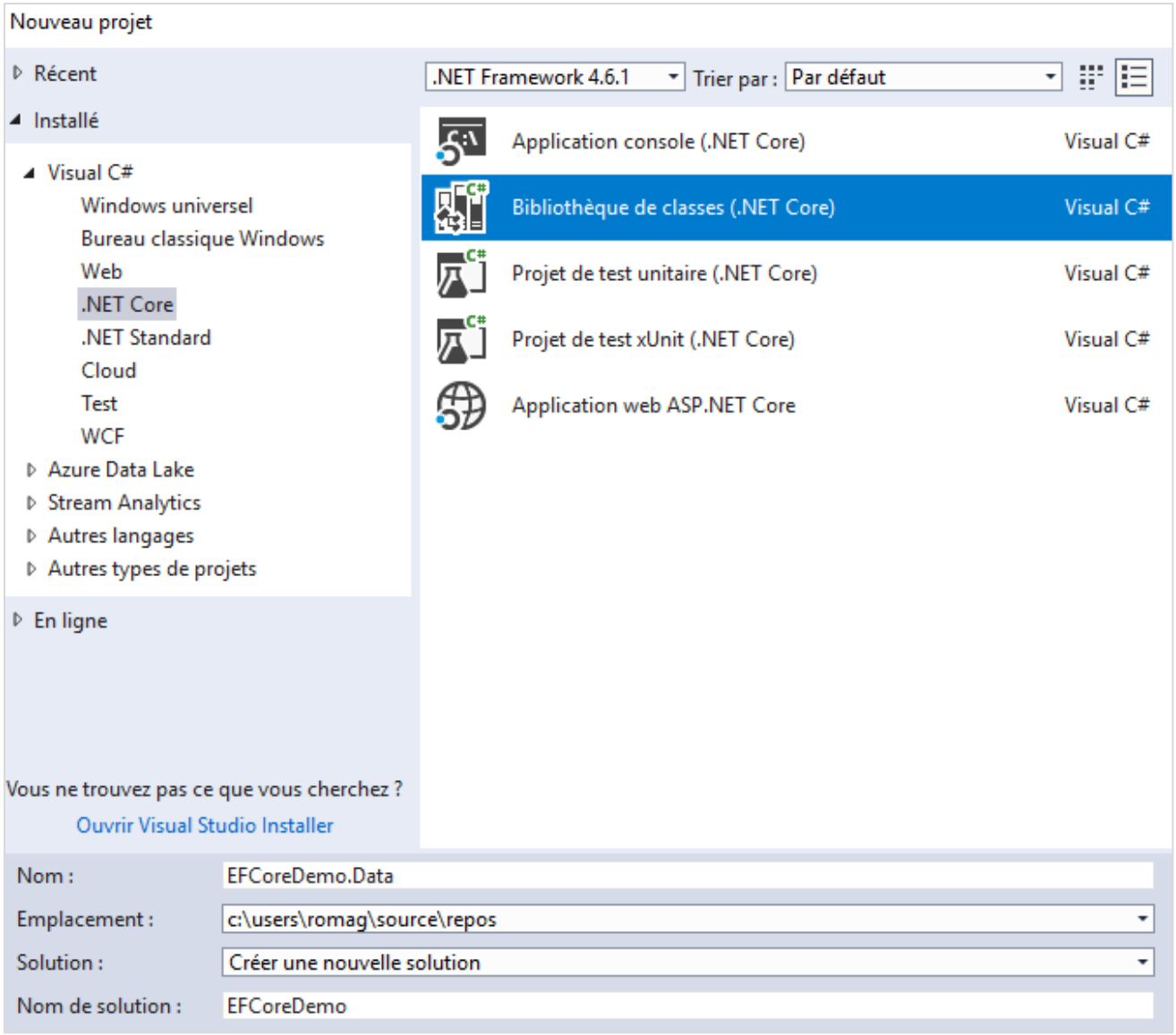
Ouvrir une **invite de commande** et saisir cette commande par exemple pour voir si l'installation s'est bien passée

```
dotnet --version
```

2. Types d'applications

a. Librairie .Net Core

On peut la nommer par exemple « EFCoreDemo.Data »



Ajout des Packages NuGet avec le **gestionnaire de packages NuGet**

- « **Microsoft.EntityFrameworkCore.SqlServer** » (il existe d'autres providers)
- Et « **Microsoft.EntityFrameworkCore.Tools** » (pour la génération de fichier de migrations

Ou depuis la **Console de Gestionnaire de package** (menu « Affichage » … « Autres fenêtres » … « Console de Gestionnaire de package »)

```
PM> Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Et

```
PM> Install-Package Microsoft.EntityFrameworkCore.Tools
```

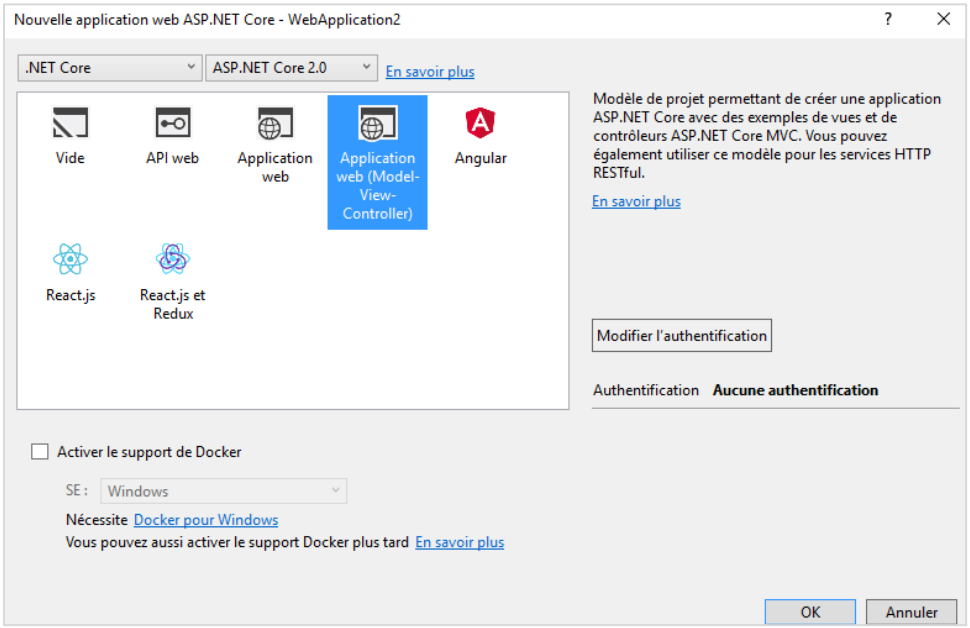
Création du DBContext

Chaine de connexion définie dans « OnConfiguring » (exemple création de la base blog)

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

namespace EFCoreDemo.Data
{
    public class Post
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }
    }

    public class BlogContext : DbContext
    {
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=blog;Integrated Security=True;");
        }
    }
}
```

b. Application Web Asp.Net Core

Créer un Projet onglet « .Net Core » … « Application Web Asp.Net Core »



Entity Framework est déjà installé avec le package NuGet « Microsoft.AspNetCore.All »

Configuration

Dans « Startup.cs ». Enregistrement du dbContext dans les services

```
public void ConfigureServices(IServiceCollection services)
{
            services.AddDbContext<MyContext>(options =>
            {
                options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));
            });

            services.AddMvc();
}
```

Ajouter la chaine de connexion dans « appsettings.json »

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=iddemo;Integrated
Security=True;"
  }
}
```

Il est possible de créer un fichier de configuration (« config.json » par exemple) à la racine du projet avec la chaine de connexion.

Dans « Program.cs »

```
public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .ConfigureAppConfiguration(SetupConfiguration)
                .UseStartup<Startup>()
                .Build();

private static void SetupConfiguration(WebHostBuilderContext ctx, IConfigurationBuilder builder)
{
            builder.Sources.Clear();
            builder.AddJsonFile("config.json", false, true);
}
```

Création des entités et dbContext

Point particulier : DbContext avec Authentification et autorisation

- Faire hériter de IdentityDbContext
- Ajouter un constructeur avec les options

```
public class MyContext: IdentityDbContext<MyUser>
{
        // db sets ...

        public MyContext(DbContextOptions<MyContext> options):base(options)
        {

        }
}
```

Création d'une classe User » héritant de « IdentityUser » (à laquelle on peut ajouter des propriétés supplémentaires)

```
  public class MyUser : IdentityUser
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
```

c. Uwp

Sélectionner la version « 16299 » minimale.

Ajouter le package NuGet « Microsoft.EntityFrameworkCore.Sqlite » avec le Gestionnaire de package ou la console du Gestionnaire de package

```
PM> Install-Package Microsoft.EntityFrameworkCore.Sqlite
```

Créer le dbContext

```
using Microsoft.EntityFrameworkCore;

namespace EFCoreUwp
{
    public class Post
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }
    }

    public class BlogContext : DbContext
    {
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite(@"Data Source=blog.db");
        }
    }
}
```

Dans « App.cs »

```
public App()
{
            this.InitializeComponent();
            this.Suspending += OnSuspending;
            using (var context = new BlogContext())
            {
                context.Database.Migrate();
            }
}
```

3. Aide aux commandes PMC

Pmc-commands

Pour avoir l'aide sur les commandes

```
PM> get-help entityframeworkcore
```

Sur une commande

```
PM> get-help Add-Migration
```

4. Configuration

a. Data Annotations

Attributs

Il est possible également d'utiliser Fluent pour configurer.

Exemple :

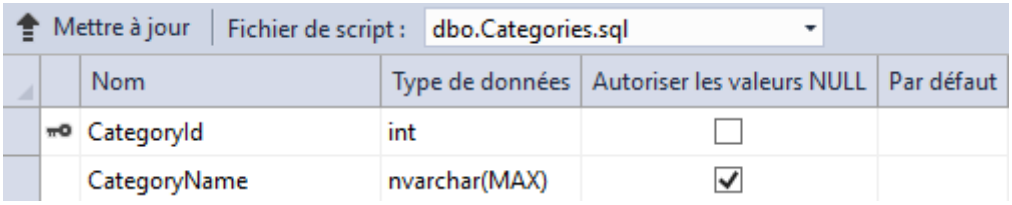
La clé primaire est par défaut « Id » (et auto-incrémentée si de type « int »). Autrement on peut utiliser l'attribut « Key » :

```
using System.ComponentModel.DataAnnotations;

namespace EFCoreDemo.Data
{
    public class Category
    {
        [Key]
        public int CategoryId { get; set; }

        public string CategoryName { get; set; }
    }
}
```

b. Fluent

Ou avec **Fluent** (dans « **OnModelCreating** »)

```
modelBuilder.Entity<Category>()
                .HasKey(category => category.CategoryId);
```



5. Relations

On définit les relations dans « OnModelCreating » .

On utilise « Include » (et « ThenInclude ») pour charger les données

One-to-one

```
    public class Person
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        public Address Address { get; set; }
    }

    public class Address
    {
        public int Id { get; set; }
        public string StreetNumber { get; set; }
```

```
        public string StreetName { get; set; }
        public string ZipCode { get; set; }
        public string City { get; set; }

        public int PersonId { get; set; }
        public Person Person { get; set; }
    }
```

Contexte

```
public class MyContext : DbContext
{
        public DbSet<Person> People { get; set; }

        // etc.
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Person>()
                .HasOne(person => person.Address)
                .WithOne(address => address.Person)
                .HasForeignKey<Address>(address => address.PersonId);
        }
}
```

Utilisation (application console .Net core par exemple)

```
using (var context = new MyContext())
{
                var people = context.People
                    .Include(person => person.Address)
                    .ToList();

                Console.ReadLine();
}
```

One-to-many

```
    public class User
    {
        public int Id { get; set; }
        public string UserName { get; set; }

        public List<Post> Posts { get; set; }
    }

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

        public int UserId { get; set; }
        public User User { get; set; }
    }
```

Contexte

```
public class BlogContext : DbContext
{
        public DbSet<User> Users { get; set; }

        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=blog;Integrated Security=True;");
            // base.OnConfiguring(optionsBuilder);
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Post>()
                .HasOne(post => post.User)
                .WithMany(user => user.Posts)
                .HasForeignKey(post => post.UserId);
        }
}
```

Utilisation

```
using (var context = new BlogContext())
{
                var posts = context.Posts.Include(post => post.User).ToList();

                var users = context.Users.ToList();
                Console.ReadLine();
}
```

Many-to-many

```
  public class Permission
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public List<UserPermission> UserPermissions { get; set; }
    }

    public class UserPermission
    {
        public int UserId { get; set; }
        public User User { get; set; }

        public int PermissionId { get; set; }
        public Permission Permission { get; set; }
    }

    public class User
    {
        public int Id { get; set; }
        public string UserName { get; set; }

        public List<UserPermission> UserPermissions { get; set; }
    }
```

Contexte

```
public class BlogContext : DbContext
    {
        public DbSet<User> Users { get; set; }

        public DbSet<Permission> Permissions { get; set; }

        // etc.
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            // clé primarie composée
            modelBuilder.Entity<UserPermission>()
                .HasKey(u => new { u.PermissionId, u.UserId });


            modelBuilder.Entity<UserPermission>()
               .HasOne<User>(userPermission => userPermission.User)
               .WithMany(user => user.UserPermissions)
               .HasForeignKey(userPermission => userPermission.UserId);

            modelBuilder.Entity<UserPermission>()
             .HasOne<Permission>(userPermission => userPermission.Permission)
             .WithMany(permission => permission.UserPermissions)
             .HasForeignKey(userPermission => userPermission.PermissionId);
        }
    }
```

Utilisation

```
using (var context = new BlogContext())
{
                var users = context.Users
                    .Include(user => user.UserPermissions)
                    .ToList();

                Console.ReadLine();
}
```

6. Crud

Fonctions :

	Asynchrone
ToList	ToListAsync
First, FirstOrDefault	FirstAsync, FirstOrDefaultAsync
Single, SingleOrDefault	SingleAsync, SingleOrDefaultAsync
Count, LongCount	CountAsync, LongCountAsync
Min, Max	MinAsync, MaxAsync
Last, LastOrDefault	LastAsync, LastOrDefaultAsync
Average	AverageAsync
Find	FindAsync

Lecture des données

Liste

```
using EFCoreDemo.Data;
using Microsoft.EntityFrameworkCore;
```

```
using System;
using System.Linq;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var context = new BlogContext())
            {
                var posts = context.Posts.ToList();
                foreach (var post in posts)
                {
                    Console.WriteLine(post.Title);
                }
                Console.ReadLine();
            }
        }
    }
}
```

Inclure des données de relations avec **Include** (et « ThenInclude »)

```
using (var context = new BlogContext())
{
                var posts = context.Posts
                    .Include(p => p.User)
                    .ThenInclude(user => user.Role)
                    .Include(p => p.Category)
                    .ToList();

                foreach (var post in posts)
                {
                    Console.WriteLine(post.Title);
                }
                Console.ReadLine();
            }
}
```

Récupérer les données de manière **asynchrone**

```
using (var context = new BlogContext())
{
       var posts = await context.Posts.ToListAsync();
}
```

Linq

```
using (var context = new BlogContext())
{
                var posts = context.Posts.Where(post => post.Title.Contains("Post")).ToList();
}
```

Un élément

```
var post = context.Posts.FirstOrDefault();
```

Ou avec Single

```
var post = context.Posts.Single(p => p.Id == 1);
```

Chargement « Explicit »

(Jusqu'à présent on faisait du eager loading)

```
using (var context = new BlogContext())
{
                var user = context.Users.Single(p => p.Id == 1);
                // avant le user à pas de posts (null)
                context.Entry(user).Collection(u => u.Posts).Load();
                // après le user a ses posts chargés
}
```

Utiliser la fonction « Reference » à la place de « Collection » pour un seul élément.

Mise à jour de données avec SaveChanges

« Disconnected »

Obtenir l'état d'une entrée

```
using(var context = new BlogContext())
{
                var post = context.Posts.Find(2);

                var entry = context.Entry(post);
}
```

Changer l'état (Added, Modified, Deleted, Detached, Unchanged)

```
context.Entry<Post>(post).State = EntityState.Modified;
```

Il faut ensuite appeler « SaveChanges » pour reporter les changements vers la base de données.

Add

```
using (var context = new BlogContext())
{
                var user = new User { UserName = "Marie" };
                context.Users.Add(user);
                context.SaveChanges();
}
```

Ou

```
using(var context = new BlogContext())
{
                var user = new User { UserName = "Pat" };
                context.Entry(user).State = EntityState.Added;
                context.SaveChanges();
}
```

Update

```
using (var context = new BlogContext())
{
                var user = context.Users.Find(1);
                user.UserName = "Deborah";
                context.SaveChanges();
}
```

Delete

```
using (var context = new BlogContext())
{
                var user = context.Users.Find (2);
                context.Remove(user);
                context.SaveChanges();
}
```

On peut également sauvegarder de manière asynchrone avec « **SaveChangesAsync** »

ChangeTracker

Ici le user ne sera pas modifié

```
using (var context = new BlogContext())
{
                var user = context.Users.AsNoTracking().Single(p => p.Id == 1);
                user.UserName = "Marie";
                context.SaveChanges();
}
```

Ou pour tout le dbcontext

```
using (var context = new BlogContext())
{
                context.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;

                var user = context.Users.Single(p => p.Id == 1);
                user.UserName = "Marie";
                context.SaveChanges();
}
```

7. Avec le CLI

Guide .Net Core, documentation

Obtenir de l'aide pour les commandes

```
dotnet --help
```

Création d'un projet avec le CLI

Liste de tous les types de projets

```
dotnet new --help
```

```
Modèles                                           Nom court       Langue          Balises
--------------------------------------------------------------------------------------------------
Console Application                               console         [C#], F#, VB    Common/Console
Class library                                     classlib        [C#], F#, VB    Common/Library
Unit Test Project                                 mstest          [C#], F#, VB    Test/MSTest
xUnit Test Project                                xunit           [C#], F#, VB    Test/xUnit
ASP.NET Core Empty                                web             [C#], F#        Web/Empty
ASP.NET Core Web App (Model-View-Controller)      mvc             [C#], F#        Web/MVC
ASP.NET Core Web App                              razor           [C#]            Web/MVC/Razor Pages
ASP.NET Core with Angular                         angular         [C#]            Web/MVC/SPA
ASP.NET Core with React.js                        react           [C#]            Web/MVC/SPA
ASP.NET Core with React.js and Redux              reactredux      [C#]            Web/MVC/SPA
ASP.NET Core Web API                              webapi          [C#], F#        Web/WebAPI
global.json file                                  globaljson                      Config
Nuget Config                                      nugetconfig                     Config
Web Config                                        webconfig                       Config
Solution File                                     sln                             Solution
Razor Page                                        page                            Web/ASP.NET
MVC ViewImports                                   viewimports                     Web/ASP.NET
MVC ViewStart                                     viewstart                       Web/ASP.NET
```

Création d'une **solution vide**

```
dotnet new sln
```

Création des dossiers pour chaque projet

```
md EFCoreDemo.Data
md ConsoleApp
```

Création d'une **libraire .Net Core**

```
cd EFCoreDemo.Data
dotnet new classlib --framework netcoreapp2.0
```

Création d'une **application console .Net Core**

```
cd ..
cd ConsoleApp
dotnet new console --framework netcoreapp2.0
```

Ajout de **référence** (ConsoleApp référence la librairie)

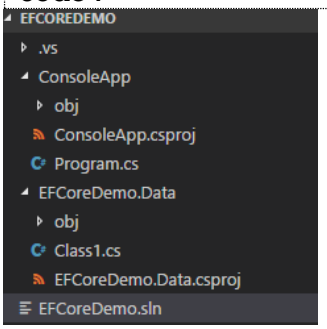
```
dotnet add reference ..\EFCoreDemo.Data\EFCoreDemo.Data.csproj
```

Ajout des **projets à la solution** (se mettre à la racine de la solution)

```
cd ..
dotnet sln EFCoreDemo.sln add ConsoleApp\ConsoleApp.csproj
EFCoreDemo.Data\EFCoreDemo.Data.csproj
```

Ouverture avec **VS Code** (« . » pour le dossier courant)

```
code .
```



Ajout de packages **NuGet**

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Tools
```

Plus …

```
dotnet add package Microsoft.EntityFrameworkCore.Tools.DotNet
```

Ajout de la section avec « DotNetCliToolReference sur « Microsoft.EntityFrameworkCore.Tools.DotNet » (En cas d'erreur « The specified framework version '2.0' could not be parsed » indiquer la version du runtime)

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
    <RuntimeFrameworkVersion>2.0.3</RuntimeFrameworkVersion>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="2.0.1" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.0.1" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.1" />
  </ItemGroup>

  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.1" />
  </ItemGroup>

</Project>
```

Puis

```
dotnet build
```

Création du **DbContext** …

Création de **migrations**

```
dotnet ef migrations add Initial
```

Mise à jour de la base de données

```
dotnet ef database update
```

Utilisation CRUD avec application console …

Lancer l'application console par exemple

```
dotnet build
dotnet run
```

Aide aux commandes

```
dotnet ef
```

```
Commands:
  database    Commands to manage the database.
  dbcontext   Commands to manage DbContext types.
  migrations  Commands to manage migrations.
```

Migrations

```
dotnet ef migrations
```

```
Commands:
  add     Adds a new migration.
  list    Lists available migrations.
  remove  Removes the last migration.
  script  Generates a SQL script from migrations.
```

Database

```
dotnet ef database
```

```
Commands:
  drop    Drops the database.
  update  Updates the database to a specified migration.
```

Rollback

Annuler une migration. Exemple

```
dotnet ef database update AddRole
```

Ou

```
dotnet ef database update 0
```

Et supprimer le dernier fichier de migration

```
dotnet ef database remove
```

Créer divers types d'applications utilisant la librairie .Net Core

Versions cibles

Editer le csproj de la librairie .Net Core, changer « TargetFramework » pour « TargetFrameworks » et ajouter à la suite par exemple « net461 »

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFrameworks>netcoreapp2.0;net461</TargetFrameworks>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="2.0.1" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.0.1" />
  </ItemGroup>

</Project>
```

8. Créer un Logger

```
using System;
using Microsoft.Extensions.Logging;

namespace EFCoreDemo.Data
{
    public class MyLoggerProvider : ILoggerProvider
    {
        public ILogger CreateLogger(string categoryName)=> new MyLogger();

        public void Dispose()
        { }

        private class MyLogger : ILogger
        {
            public bool IsEnabled(LogLevel logLevel) => true;

            public void Log<TState>(LogLevel logLevel, EventId eventId, TState state,
Exception exception,
              Func<TState, Exception, string> formatter)
            {
                // exemple filtre sur les informations (sql queries)
                if (logLevel == LogLevel.Information)
                {
                    Console.WriteLine(formatter(state, exception));
                    Console.WriteLine();
                }
            }

            public IDisposable BeginScope<TState>(TState state)
            {
                return null;
            }
        }
    }
}
```

Utilisation

```
using EfConflict.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.Extensions.Logging;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var context = new BlogContext())
            {
                context.GetService<ILoggerFactory>().AddProvider(new MyLoggerProvider());
                // etc.
                Console.ReadLine();
            }
        }
    }
}
```

9. Unit Of Work

Exemple avec Asp.Net Core

Interface « IUnitOfWork »

```
using System.Threading.Tasks;

namespace WebApp.Core
{
    public interface IUnitOfWork
    {
        Task CompleteAsync();
    }
}
```

« UnitOfWork »

```
using System.Threading.Tasks;
using WebApp.Core;

namespace WebApp.Persistence
{
    public class UnitOfWork : IUnitOfWork
    {
        private BlogContext _context;

        public UnitOfWork(BlogContext context)
        {
            _context = context;
        }

        public async Task CompleteAsync()
        {
           await _context.SaveChangesAsync();
        }
    }
}
```

Repository

```
public interface IPostRepository
{
        Task<List<Post>> GetAllAsync();
        void Add(Post post);
        void Remove(Post post);
}
```

```
public class PostRepository : IPostRepository
{
        private BlogContext _context;

        public PostRepository(BlogContext context)
        {
            _context = context;
        }

        public async Task<List<Post>> GetAllAsync()
        {
            return await _context.Posts.ToListAsync();
        }

        public void Add(Post post)
        {
            _context.Posts.Add(post);
        }

        public void Remove(Post post)
        {
            _context.Posts.Remove(post);
        }
}
```

Pas de « SaveChanges », c'est l'unit of work qui s'en occupe

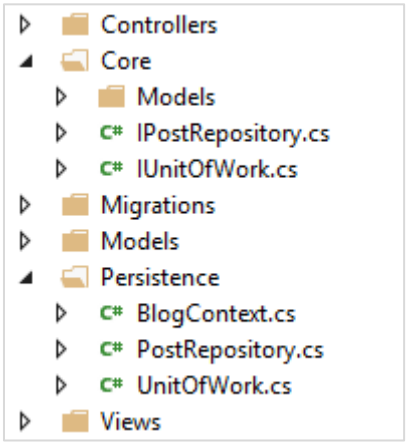
Enregistrement dans les service (« Startup.cs »)

```
public void ConfigureServices(IServiceCollection services)
{
            services.AddScoped<IPostRepository, PostRepository>();
            services.AddScoped<IUnitOfWork, UnitOfWork>();

            services.AddDbContext<BlogContext>(options =>
            {
                options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));
            });

            services.AddMvc();
}
```

Création d'un contrôleur

```
[Route("/api/posts")]
public class PostsController : Controller
{
        private readonly IPostRepository repository;
        private readonly IUnitOfWork unitOfWork;

        public PostsController(IPostRepository repository, IUnitOfWork unitOfWork)
        {
```

```
            this.unitOfWork = unitOfWork;
            this.repository = repository;
        }


        [HttpPost]
        public async Task<IActionResult> CreatePost([FromBody] Post post)
        {
            repository.Add(post);
            // autres opérations
            await unitOfWork.CompleteAsync();

            return Ok(post);
        }
}
```



- **Core** : abstractions, Models
- **Persistence** : unit of work, contexte et repositories