ASP.NET Core : Scaffolding (Web API + EF)

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

ASP.NET Core

Objectif

- Utilisation d'**Entity Framework**
- Génération d'un contrôleur **Web API** avec actions utilisant **Entity Framework**.

ASP.NET Core

Objectif

- Utilisation d'**Entity Framework**
- Génération d'un contrôleur **Web API** avec actions utilisant **Entity Framework**.

Pour la suite

- Créons un nouveau projet `CoursWebApiEF` dans la solution `CoursAspNetCore`.
- Copions tous les répertoires, sauf `Controllers`, du projet `CoursWebApi` dans le nouveau projet `CoursWebApiEF`.
- Copions également le contenu de `Program.cs`.

ASP.NET Core

Contenu à garder de `Program.cs`

```
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

ASP.NET Core

Packages à installer avec **NuGet**

- **Microsoft.EntityframeworkCore.SqlServer** qui inclut déjà **Microsoft.EntityFrameworkCore**
- **Microsoft.EntityFrameworkCore.Tools** pour gérer les migrations

ASP.NET Core

Dans `Configs`**, créons une classe** `ApplicationContext` **qui hérite de** `DbContext`

```
using Microsoft.EntityFrameworkCore;

namespace CoursWebApiEF.Configs
{
    public class ApplicationContext : DbContext
    {
    }
}
```

ASP.NET Core

Dans `ApplicationContext`, définissons le constructeur avec paramètre suivant (indispensable pour l'injection de dépendances)

```
using Microsoft.EntityFrameworkCore;

namespace CoursWebApiEF.Configs
{
    public class ApplicationContext : DbContext
    {
        public ApplicationContext(DbContextOptions<ApplicationContext> options) : base(options)
        {
        }
    }
}
```

Dans `Program.cs`, spécifions la chaîne de connexion et définissons `ApplicationContext` comme service

```
using CoursWebApiEF.Configs;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();


builder.Services.AddDbContext<ApplicationContext>(options =>
    options.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=FormationWS;
      Trusted_Connection=True;")
);

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

ASP.NET Core

On peut aussi définir la chaîne de connexion dans `appsettings.json`

```
"ConnectionStrings": {
    "CoursWebApiEFContext": "Server=(localdb)\\mssqllocaldb;Database=
      FormationWS;Trusted_Connection=True;"
}
```

ASP.NET Core

On peut aussi définir la chaîne de connexion dans `appsettings.json`

```
"ConnectionStrings": {
    "CoursWebApiEFContext": "Server=(localdb)\\mssqllocaldb;Database=
      FormationWS;Trusted_Connection=True;"
}
```

Chargement de la chaîne de connexion dans `Program.cs`

```
builder.Services.AddDbContext<CoursWebApiEFContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("
      CoursWebApiEFContext")));
```

ASP.NET Core

Nous pourrons aussi alléger le contenu de `Program.cs` en déplaçant les données le code précédent dans `ApplicationContext`

```
using Microsoft.EntityFrameworkCore;

namespace CoursWebApiEF.Configs
{
    public class ApplicationContext : DbContext
    {
        private readonly IConfiguration _configuration;

        public ApplicationContext(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        protected override void OnConfiguring(DbContextOptionsBuilder options)
        {
            options.UseSqlServer(Configuration.GetConnectionString("CoursWebApiEFContext"));
        }

    }
}
```

ASP.NET Core

Nous pourrons aussi alléger le contenu de `Program.cs` en déplaçant les données le code précédent dans `ApplicationContext`

```
using Microsoft.EntityFrameworkCore;

namespace CoursWebApiEF.Configs
{
    public class ApplicationContext : DbContext
    {
        private readonly IConfiguration _configuration;

        public ApplicationContext(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        protected override void OnConfiguring(DbContextOptionsBuilder options)
        {
            options.UseSqlServer(Configuration.GetConnectionString("CoursWebApiEFContext"));
        }

    }
}
```

Dans `Program.cs` il reste à déclarer cette classe comme service

```
builder.Services.AddDbContext<ApplicationContext>();
```

ASP.NET Core

Modifions la classe `Personne` pour quelle soit considérée comme entité

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace CoursWebApiEF.Models
{
    public class Personne
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Num { get; set; }

        [Required]
        public string? Nom { get; set; }

        [StringLength(20)]
        public string? Prenom { get; set; }

        [Range(0, 150)]
        public int Age { get; set; }

        public ICollection<Adresse>? Adresses { get; set; }
    }
}
```

ASP.NET Core

Contenu de la classe `Adresse`

```
using System.Text.Json.Serialization;

namespace CoursWebApiEF.Models
{
    public class Adresse
    {
        public int Id { get; set; }
        public string? Rue { get; set; }
        public string? Ville { get; set; }
        public string? CodePostal { get; set; }

        [JsonIgnore]
        public ICollection<Personne>? Personnes { get; set; }
    }
}
```

ASP.NET Core

Mettons à jour la classe `ApplicationContext.cs`

```
public class ApplicationContext : DbContext
{
    public DbSet<Personne> Personnes { get; set; }
    public DbSet<Adresse> Adresses { get; set; }

    // + méthodes précédentes

}
```

ASP.NET Core

Créons une première migration

```
Add-Migration FirstMigration
```

ASP.NET Core

Créons une première migration

```
Add-Migration FirstMigration
```

Exécutons la migration

```
Update-Database
```

ASP.NET Core

En cas de problème avec la mise à jour de la base de données, il faut double cliquer sur le projet et mettre à `false` la propriété suivante

```
<InvariantGlobalization>false</InvariantGlobalization>
```

ASP.NET Core

En cas de problème avec la mise à jour de la base de données, il faut double cliquer sur le projet et mettre à `false` la propriété suivante

```
<InvariantGlobalization>false</InvariantGlobalization>
```


Remarque

- Si `<InvariantGlobalization>` est défini à `false`, alors l'application utilisera la culture actuelle du système pour les opérations de globalisation : formatage des dates, des nombres...
- Si `<InvariantGlobalization>` est défini sur `true`, alors l'application utilisera la culture invariante : une culture neutre qui ne dépend pas des paramètres régionaux spécifiques du système.

ASP.NET Core

Pour générer le code d'un contrôleur **Web API** utilisant **EF Core**, il faut

- Faire clic droit sur le nom du projet dans l'`Explorateur de solutions` et choisir `Générer`
- Faire clic droit sur le répertoire `Controllers` et aller dans `Ajouter > Contrôleur`
- Choisir `Contrôleur d'API avec actions, utilisant Entity Framework`
- Dans `Classe de modèle` sélectionner `Personne` et ajouter une nouvelle classe de contexte
- Garder le nom `PersonnesController` pour le contrôleur puis valider

ASP.NET Core

Lancer le projet et tester les différentes actions de notre contrôleur `PersonnesController`.