TP2 : EF Core & CRUD Operations

A. Création des classes d'entités (Model)

1. Définir les classes d'entités suivantes sans relations : Movie, Genre & Customer. Chaque classe a deux propriétés (Id & Name)

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

public class Genre
    {
        public Guid Id { get; set; }
        public string Name { get; set; }
    }
public class Movie
    {
        public  int Id { get; set; }
        public string? Name { get; set; }
        public Movie()
        {

        }
    }
```

B. Etapes de migration EF Core 6 pour la génération automatique de la base de données « approche CodeFirst »

Le Microsoft Entity Framework Core est un Framework de mappage objet/Databases (ORM) qui permet aux développeurs de travailler avec des données relationnelles en tant qu'objets, éliminant ainsi le besoin de la plupart du code de plomberie d'accès aux données que les développeurs doivent généralement écrire. À l'aide d'Entity Framework, les développeurs émettent des requêtes à l'aide de LINQ (Language Integrated Query), puis récupèrent et manipulent les données en tant qu'objets fortement typés.

Deux approches de modélisation pour Entity Framework Core:

1. Code First : Dans l'approche Code First, on crée tout d'abord nos classes par la suite on passe à concevoir notre base de données à partir des classes déjà construites.

2. Database First : ça consiste à générer les classes et les relations entre les classes à partir d'une base de données existante.

- Commencer par installer les packages suivants pour entity framework Core, via PM> ou CLI> ouinterface NuGet Packages:
  - Microsoft.EntityFrameworkCore
  - Microsoft.EntityFrameworkCore.Design
  - Microsoft.EntityFrameworkCore.SqlServer

- Microsoft.EntityFrameworkCore.Tools

PS : Vous pouvez installer ces packages, de même, via **la console de gestionnaire de package** (avec la commande ***install-package nom-package***) ou bien via **le power shell** (avec la commande ***dotnet add package nom-package***).

- **Consulter** le fichier NomProjet.csproj pour vérifier que les dépendances (packages) ontbien été rajoutées :

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

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="6.0.22" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design"
Version="6.0.22">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer"
Version="6.0.22" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools"
Version="6.0.22">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design"
Version="6.0.16" />
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.5.0" />
  </ItemGroup>

</Project>
```

- Aller maintenant dans le fichier Appsettings.json et configurer la chaîne de connexionà la base de données.

```
"ConnectionStrings": {
    "DefaultConnection": "Data Source=(localdb)\\mssqllocaldb ; database =
TestDatabaseConnection ; Integrated Security=True"

  },
```

- Rajouter une classe de contexte « ApplicationDbContext.cs » sous le dossier Models pour la configuration de la connexion avec la base de données. Cette classe est considérée comme une passerelle entre la partie métier de l'application et la base de données. **C'est là où on doit spécifier les options de connexion ainsi que les DbSet à mapper.**
- ```
  public class ApplicationdbContext : DbContext
   {
       public ApplicationdbContext(DbContextOptions options)
           :base(options)
       {

       }
  ```
- Activer le service relatif au DbContext au niveau du conteneur de service dans le fichier program.cs.
  ```

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

PS : N'oubliez pas d'importer les packages manquants

- Définir les ensembles de données dans la classe du contexte ApplicationDbContext (DbSet) pour être générés au niveau de la base de données.

```
public DbSet<Movie>? movies { get; set; }
public DbSet<Genre> genres { get; set; }
```

- Ajouter les migrations nécessaires et faites une mise à jour de la base de données. Par la suite consulter votre base de données à partir de l'explorateur du serveur.

C. Création des contrôleurs et des vues

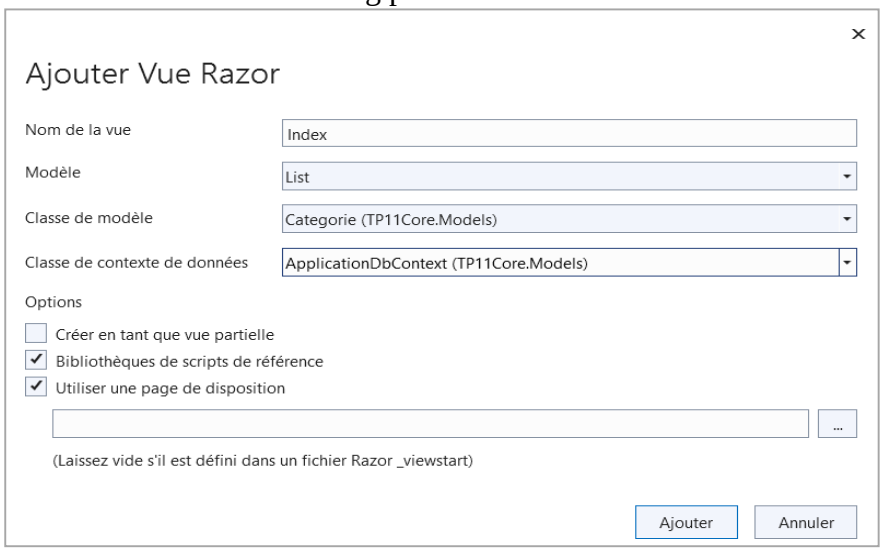
- Générer l'application, et créer par la suite le contrôleur CategorieController etInjecter explicitement ApplicationDbContext dans le constructeur.

```
private readonly ApplicationdbContext _db;
public GenreController(ApplicationdbContext db)
{
    _db = db;
}
```

- Changer l'action « Index » de MovieController pour lister à partir de la base de données de la base.

```
public IActionResult Index()
        {
            var movies = _db.movies.ToList();
            return View(movies);
        }
```

- Générer la vue Index correspondante fortement typée automatiquement en utilisantle scaffolding pour un modèle list.



- Créer l'action « Create » permettant de créer un nouveau film au niveau de la base. Mettez la validation Forgery pour prevenir les attaques XSS ➔ pour ne pas écrire un script au niveau des champs

```
public IActionResult Create()
```

```
{
    return View();
}
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(Movie movie)
{
    _db.genres.Add(movie);
    _db.SaveChanges();
    return RedirectToAction(nameof(Index));
}
```

- Générer la vue « Create.cshtml » correspondante fortement typée automatiquementen utilisant le scaffolding pour un modèle Create.

C'est à vous :

- Créer l'action « Edit» permettant de mettre à jour un film au niveau de la base.
- Créer l'action « Delete » pour supprimer une de la base de données
- Générer la vue « Edit.cshtml » correspondante fortement typée automatiquement en utilisant le scaffolding pour un modèle Edit.
- Générer la vue « Delete.cshtml » correspondante fortement typée automatiquement en utilisant le scaffolding pour un modèle Delete, en changeant le nom de l'action du formulaire.
- Chaque film appartient à un Genre. Chaque Genre peut contenir plusieurs Films : Définir la relation suivante en Code First.

D. TP à faire chez vous

- Elaborez les étapes de connexion que vous juges pertinentes dans une approche Database First. Veuillez fournir un compte rendu décrivant les étapes de réalisation. **Pensez à utiliser dotnet-ef dbcontext scaffold**