**1 - ACTIONRESULT**

ActionResult est un type de retour utilisé dans les actions des contrôleurs ASP.net Core.

C’est un type générique qui encapsule le résultat d’une action et retourne des résultats http tels que ‘OK’, ‘BadRequest’, ‘NotFound’

**2 - Différence entre IEnumerate et List :**

List est une classe qui implémente IEnumerate. Elle offre, en plus de sa feature d’itération, des méthodes supplémentaires pour manipuler des listes telles que « Add », « Remove » etc…

Un contrôleur « getAll » renvoie donc une ActionResult de IEnumerable d’objets LUTINS DTO.

A l’intérieur du contrôlleur,, le mapper se charge de mapper des Objets Lutins récupérés en Lutins DTO, se sont ces lutins DTO qui seront renvoyés lors de la requête http.

L’utilité des DTOS ? Avoir un contôle plus précis sur les données à renvoyer, comme par exemple limiter le nombre de propriétés.

**3 – Base to Model :**

scaffold-DbContext -Connection "Server=localhost;Database=personnesdb;user=root;password=" -Provider mysql.entityFrameworkCore -OutputDir Models/Data -Context PersonnesDbContext

Permet de récupérer les informations en base de données et de construire le DbContext + les classes correspondants aux tables.

Afin de dissocier le context de data, et de la mettre à la racine :

scaffold-DbContext -Connection name=Default MySql.EntityFrameworkCore -outputDir ".\Models\Data" -Context "nppContext" -ContextDir ".\Models"

* Si la connexion string est configurée dans appsettings.json, utiliser :

scaffold-DbContext -Connection Name=Default MySql.EntityFrameworkCore -outputDir .\Models\DbModels

appsetiings :

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"Default": "server=localhost;user=root;database=npp;port=3306;"

}

}

**4 – Ensuite avant d’effectuer une première migration il faut ajouter le contexte en tant que service dans le Program.cs, avec les options de configuration :**

builder.Services.AddDbContext<NppContext>(options => options.UseMySQL(builder.Configuration.GetConnectionString("Default")));

Puis lancer la migration avec add-migration FirstMigration. Cela lancera le snapShot global de l’état de la base de donnée.

Ensuite supprimer la FirstMigration, et le tour et joué, les autres migrations seront valides. Attention pour une propriété « Id » il faut réellement mettre « Id ».

**5 – Dans le Programs.cs, il faut définir les services , avec possibilité d’utiliser les nuggets en tant que services :**

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddDbContext<ApiDbContext>(options => options.UseMySQL(builder.Configuration.GetConnectionString("Default")));

builder.Services.AddControllersWithViews();

builder.Services.AddTransient<LutinsServices>();

builder.Services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

builder.Services.AddControllers().AddNewtonsoftJson();

var app = builder.Build();

Addtransient : Un service transitoire signifie que chaque fois qu’une classe demande une instance de ‘ LutinsService’, le conteneur d’injection de dépendance en crée une nouvelle.

**5 – Migrations :**

Créer une migration : Add-migration <Nom de la migration>

Si dans le terminal et non dans le gestionnaire de package, ajouter « dotnet ef »

Supprimer migrations : remove-migration

Mettre à jour la base de données : update-database

**6 La liste des dépendances à utiliser :**

  <ItemGroup>

    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="7.0.13" />

    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.5.0" />

  </ItemGroup>

  <ItemGroup>

    <PackageReference Include="AutoMapper" Version="12.0.1" />

    <PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="12.0.1" />

    <PackageReference Include="Microsoft.AspNetCore.JsonPatch" Version="8.0.0" />

    <PackageReference Include="Microsoft.AspNetCore.Mvc.NewtonsoftJson" Version="7.0.14" />

    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="7.0.14" />

    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="7.0.14">

      <PrivateAssets>all</PrivateAssets>

      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>

    </PackageReference>

    <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="7.0.11" />

    <PackageReference Include="MySql.Data" Version="8.2.0" />

    <PackageReference Include="MySql.EntityFrameworkCore" Version="7.0.10" />

  </ItemGroup>

**7 - TODO list**

1 - Créer un nouveau projet API ASP.NET

2 – Modifier appsetting.json pour y inscrire la connexionString

3 – Mettre les dépendances et régénérer le projet

4 – Scaffffffffold !!!

4 – Changer le nom des variables des sytèmes relationnels. ( ctrl-h)

5 – Déclarer le contexte dans le program.cs

6 – Déclarer les services – ajout dans Program – mettre les includes.

7 – Faire les DTOs : Globaux, In et Out

8 – Créer les profils

**8- Exemple ObjectHelper :**

    static class ObjectHelper

    {

        public static void Dump(this object data)

        {

            string json = JsonConvert.SerializeObject(data, Newtonsoft.Json.Formatting.Indented);

            Trace.WriteLine("");

            Trace.WriteLine(json);

            // installation, ajouter les packages :

            //     < PackageReference Include = "Microsoft.AspNetCore.JsonPatch" Version = "5.0.10" />

            //     < PackageReference Include = "Microsoft.AspNetCore.Mvc.NewtonsoftJson" Version = "5.0.10" />

            // ajouter la configuration dans startup

            //      services.AddControllers().AddNewtonsoftJson(s => {

            //      s.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();});

            //Utilisation   objet.Dump();

        }

    }

**9 - Points d’attentions :**

* Avant de renommer le nom des relations, penser à vérifier où les noms peuvent être modifiés dans le projet ( en général dans le contexte dans les relations ( with many ).
* Attention, on garde les Ids dans les DTO out et non dans les DTO in !
* Les DTO « In » servent à create et put, les DTO « out » servent à tout ce qui touche à la récupération de datas.
* Si la classe est « out », alors dans les relations on aurait forcément des DTOs out.
* Dans les includes, ne pas oublier de mettre le nom des relations définies dans les méthodes virtuelles.
* Si les includes vont dans le même sens, exemple clients->commandes->articles alors on utilise un seul include : client Include(commandes.articles), sinon deux includes.
* Si virtual ICollection -> liste <---->
* Si Virtual Entité -> Le <--->
* Les ForMembers servent uniquement pour les applatissements.
* Ne pas hésiter à commit régulièrement.
* Mapper DANS les controllers en fonction des DTOs utilisés.