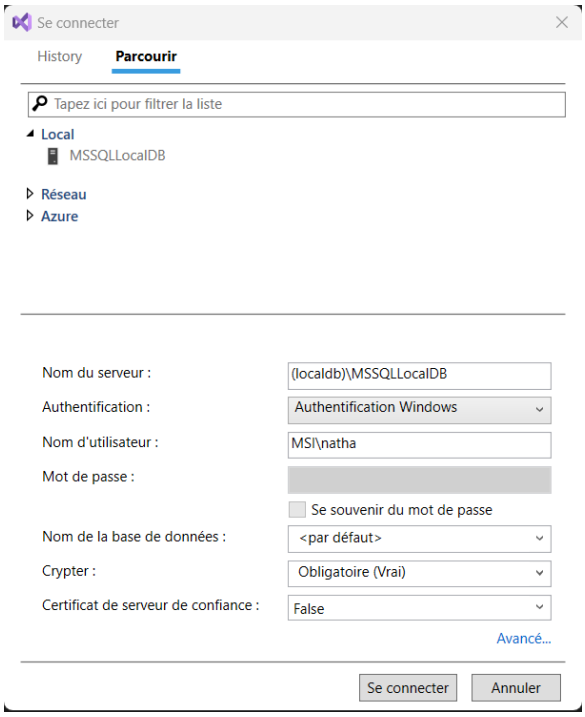
Exam DotNet

Base de données :

1. Créer la base de données Northwind à partir du fichier « Northwind4SqlServer.sql»
    a. Récupérer sur MooVin le fichier « Northwind4SQLServer.sql» dans les ressources
    b. Exécuter un query avec ce fichier SQL dans Visual Studio
        i. Outils –> SQL SERVER -> Nouvelle requête/query
        ii. Copier-coller le code du fichier «Northwind4SQLServer.sql»
        iii. Exécuter le code du query (bouton play vert en haut à gauche de la fenêtre query)

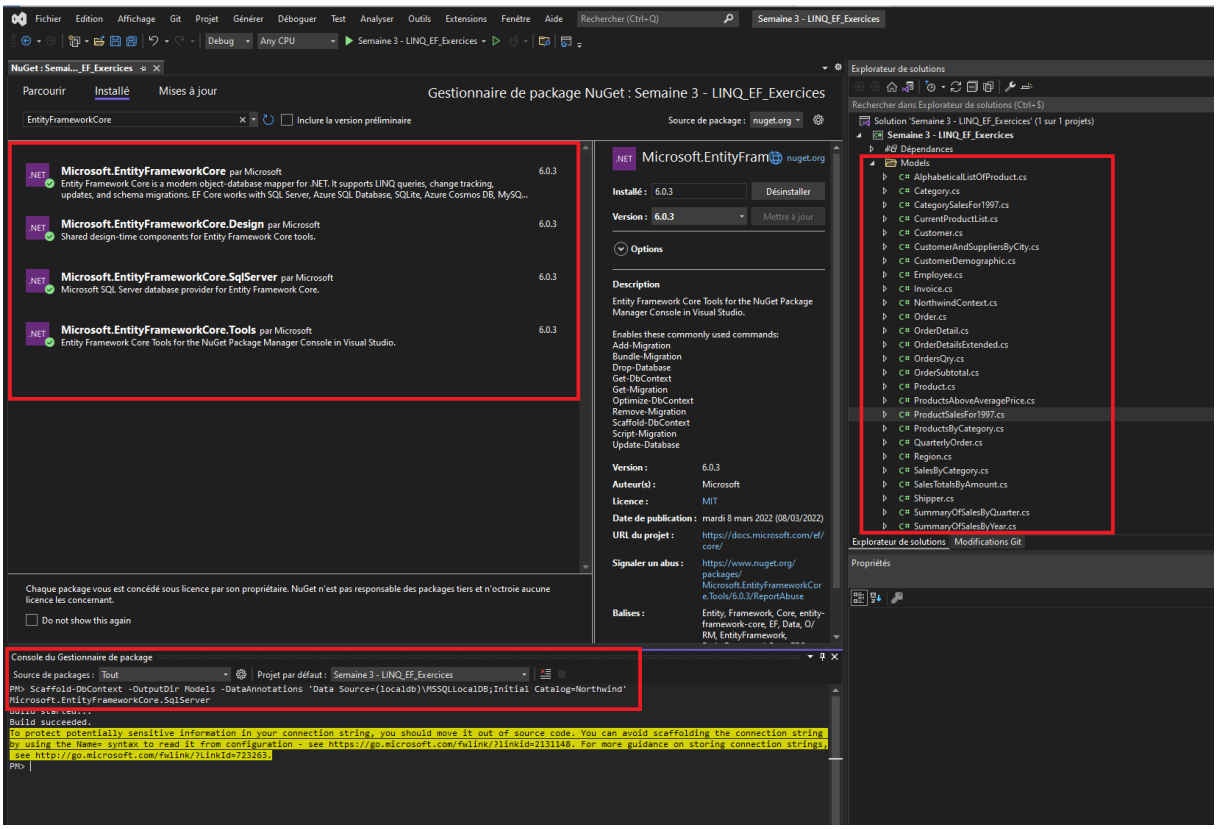


2. Voir la base de données Northwind
    a. Dans Visual Studio, vous pouvez manipuler les bases de données SQL SERVER via l'explorateur d'objets SQL SERVER
    b. Affichage -> Explorateur d'objets SQL SERVER
    c. Vous devez vous connecter à une instance SQL Server. Nous utiliserons toujours l'instance : **(localdb)\MSSQLLocaldb**
    d. Vérifier que vous avez bien une base de données Northwind et afficher les données d'une table

3. Ajouter les packages NuGet nécessaires pour EntityFramework
    a. Clic droit sur le projet -> Gérer les packages NuGet
    b. Ajouter les packages
        i. Microsoft.EntityFrameworkCore
        ii. Microsoft.EntityFrameworkCore.Design
        iii. Microsoft.EntityFrameworkCore.Tools
        iv. Microsoft.EntityFrameworkCore.SqlServer
4. Créer les classes-entités depuis la DB
    a. **Dans une console du Package Manager (PM)**
        i. Outils – Gestionnaire de Package NuGet -> Console
    b. Lancez la commande (sur une seule ligne) :
5. **Commande :**

Scaffold-DbContext -OutputDir Models 'Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Northwind' Microsoft.EntityFrameworkCore.SqlServer

En image :



Activer le lazy loading avec utilisation d'un proxy

a. Installer le package EntityFrameworkCore.Proxies

b. Aller dans votre fichier DbContext Nortwind généré lors de l'étape A.

c. Modifier la méthode OnConfiguring comme ceci :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
if (!optionsBuilder.IsConfigured)
{
    optionsBuilder.UseSqlServer("Data Source=(localdb)\\MSSQLLocalDB;Initial
                Catalog=Northwind;MultipleActiveResultSets=True")
        .UseLazyLoadingProxies()
        .LogTo(Console.WriteLine, LogLevel.Information)
        .EnableSensitiveDataLogging();
        }
}
```

using Microsoft.Extensions.Logging;

App.xaml :

Uri = Views/MainWindow.xaml

M = Models (Entity)

VM = ViewModels (EntityModel, EntityVM)

V = Views (MainWindow.xaml)

MainWindow.xaml.cs :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using WpfEmployee.ViewModels;

namespace WpfEmployee
{
```

```
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new EntityVM();
        }
    }
}
```

Liaison entre le CS et le XAML.

Analyse des éléments de la classe

1. **public partial class MainWindow : Window** :
    - La classe **MainWindow** hérite de la classe **Window**, ce qui signifie qu'elle représente une fenêtre dans l'interface utilisateur de l'application.
2. **public MainWindow()** :
    - Le constructeur de la classe.
    - Il initialise la fenêtre et ses composants grâce à la méthode **InitializeComponent()**. Cette méthode est générée automatiquement par le fichier XAML associé (généralement MainWindow.xaml) et configure tous les éléments définis dans ce fichier.
3. **this.DataContext = new EntityVM();** :
    - Associe un objet de type **EntityVM** (ViewModel) à la propriété **DataContext** de la fenêtre.
    - Le **DataContext** est utilisé pour établir une liaison entre les éléments de l'interface utilisateur (définis dans le XAML) et les données ou la logique du ViewModel. Cela permet de connecter dynamiquement les

propriétés du ViewModel aux contrôles de la fenêtre en utilisant le **data binding**.

Rôle global de cette classe

1. **Initialisation de l'interface utilisateur** :
    - La méthode **InitializeComponent()** configure l'interface définie dans le fichier XAML associé.
2. **Connexion du ViewModel** :
    - En définissant **this.DataContext = new EntityVM();**, la classe établit une connexion avec le ViewModel, permettant aux contrôles de la fenêtre d'accéder aux propriétés, méthodes et commandes définies dans **EntityVM**.
3. **Point d'entrée pour les interactions utilisateur** :
    - C'est ici que la logique spécifique à la fenêtre peut être ajoutée, comme les gestionnaires d'événements pour les boutons, les actions spécifiques, ou les interactions utilisateur.

En résumé

Cette classe :

- Charge et initialise l'interface utilisateur définie dans le fichier **MainWindow.xaml**.
- Connecte la fenêtre à un ViewModel (**EntityVM**) en définissant son **DataContext**, permettant ainsi le **data binding** entre la vue et les données.

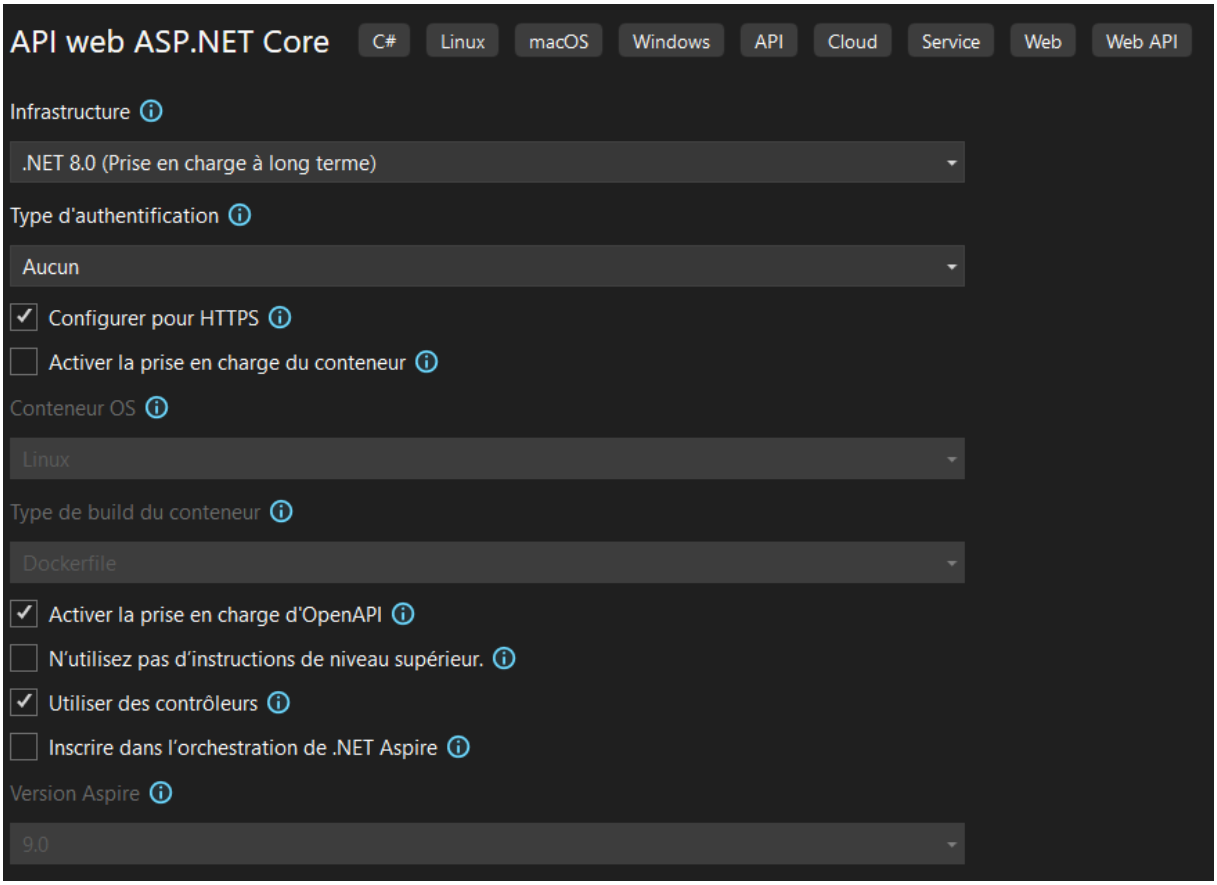
C'est un exemple typique d'implémentation du modèle MVVM (Model-View-ViewModel) dans une application WPF.

ItemsSource="{Binding StudentList}"

ItemTemplate="{StaticResource listTemplate}"

SelectedItem="{Binding SelectedStudent}" sélectionne l'item dans la liste

DataContext="{Binding SelectedStudent}" mets à jours dans les champs




```
public ObservableCollection<ProductsModel> ProductsList
{
    get
    {
        if (_productsList == null)
        {
            _productsList = loadProducts();
        }

        return _productsList;

    }
}
```

WPF Setup

1. Création des dossiers « MVVM »
2. Création base de données (script SQL)
3. Package Nuggets : Entities Framework
4. Exécution commande console package nuggets : **Scaffold-DbContext -OutputDir Models 'Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Northwind' Microsoft.EntityFrameworkCore.SqlServer**
5. **UseLazyLoadingProxies()** dans le “DbContext” file

WEB API Setup

1. Création dossier « Entities »
2. Création base de données (script SQL)
3. Package Nuggets : Entities Framework
4. Exécution commande console package nuggets : **Scaffold-DbContext -OutputDir Entities 'Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Northwind' Microsoft.EntityFrameworkCore.SqlServer**
5. **UseLazyLoadingProxies()** dans le “DbContext” file
6. Création Repository et UnitOfWork
7. Création DTO