Les propriétés de navigation :

```
using System;
using System.Collections.Generic;

namespace SchoolApp.Models;

public partial class Course
{
    public int CourseId { get; set; }

    public string Name { get; set; } = null!;

    public int ProfessorId { get; set; }

    public virtual Professor Professor { get; set; } = null!; // ici, c'est la prop de navigation
}
```

Repository :

Centralise l'accès aux données. Fait les opérations de CRUD.
1 interface, 1 classe qui l'implémente et qui est générique. Si un modèle a besoin de méthode spécifique (ex : getAllStudentByClass) on créer, en plus, une interface et une classe qui l'étend. Ces 2 classes devront en plus, étendre l'interface de base.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace School.Repository
{
    // interface générique pour les repositories
    public interface IRepository<T>
    {
        void Insert(T entity);
        void Delete(T entity);
        IList<T> SearchFor(Expression<Func<T, bool>> predicate);
        // sauve l'entité si l'élément n'existe pas déjà -> l'existence se base sur le prédicat
        bool Save(T entity,Expression<Func<T, bool>> predicate);
        IList<T> GetAll();
```

```
        T GetById(int id);
    }
}
using Microsoft.EntityFrameworkCore;
using SchoolApp.Models;
using System.Linq.Expressions;

namespace School.Repository
{

    public class BaseRepositorySQL<TEntity> : IRepository<TEntity> where TEntity : class
    {

        protected readonly SchoolContext _dbContext;
        public BaseRepositorySQL(SchoolContext dbContext)
        {
            _dbContext = dbContext;
        }

        public void Insert(TEntity entity)
        {

            _dbContext.Set<TEntity>().Add(entity);

            SaveChanges();
        }

        public void Delete(TEntity entity)
        {
            _dbContext.Set<TEntity>().Remove(entity);
            SaveChanges();
        }

        public IList<TEntity> SearchFor(Expression<Func<TEntity, bool>> predicate)
        {
            return _dbContext.Set<TEntity>().Where(predicate).ToList();
        }

        public IList<TEntity> GetAll()
        {
```

```
        return _dbContext.Set<TEntity>().ToList();
    }

    public TEntity GetById(int id)
    {
        return _dbContext.Set<TEntity>().Find(id);
    }

    public bool Save(TEntity entity, Expression<Func<TEntity, bool>> predicate)
    {
        TEntity ent = (SearchFor(predicate)).FirstOrDefault();

        if (ent == null)
        {
            Insert(entity);
            return true;
        }
        SaveChanges();

        return false;
    }

    protected void SaveChanges()
    {
        try
        {
            _dbContext.SaveChanges();
        }
        catch (DbUpdateException ex)
        {
            throw new DbUpdateException(ex.InnerException.Message);
        }
    }
  }
}
```

Utilisation :

```
SchoolContext context = new SchoolContext();
```

```
// création repositories
BaseRepositorySQL<Section> repoSect = new BaseRepositorySQL<Section>(context);

// ajout de 2 sections
Section sectInfo = new Section { Name = "Info" };
repoSect.Save(sectInfo, s => s.Name.Equals(sectInfo.Name));
Section sectDiet = new Section { Name = "Diet" };
repoSect.Save(sectDiet, s => s.Name.Equals(sectDiet.Name));

// renvoyer toutes les sections
IList<Section> sections = repoSect.GetAll().ToList();
Console.WriteLine("---------- SECTIONS --------------------");
foreach (Section s in sections)
{
    Console.WriteLine(s.Name);
}
Console.WriteLine("----------------------------------------");
```

Unit Of Work :

Permet de fournir des instances de chaque repositories. Gères toutes les interactions avec les repositories.

```
using School.Repository;
using SchoolApp.Models;

namespace School.UnitOfWork
{
    interface IUnitOfWorkSchool
    {
        IRepository<Section> SectionsRepository { get; }

        IRepository<Student> StudentsRepository { get; }
```

```
    }
}
```

```
using School.Repository;
using SchoolApp.Models;
using SchoolApp.Models;

namespace School.UnitOfWork
{
    class UnitOfWorkSchoolSQLServer : IUnitOfWorkSchool
    {
        private readonly SchoolContext _context;

        private BaseRepositorySQL<Student> _studentRepository;

        private BaseRepositorySQL<Section> _sectionsRepository;


        public UnitOfWorkSchoolSQLServer(SchoolContext context)
        {
            this._context = context;
            this._studentRepository = new BaseRepositorySQL<Student>(context);
            this._sectionsRepository = new BaseRepositorySQL<Section>(context);

        }

        public IRepository<Student> StudentsRepository
        {
            get { return this._studentRepository; }
        }

        public IRepository<Section> SectionsRepository
        {
            get { return this._sectionsRepository; }
        }
    }
}
```

Utilisation :

```
using School.Repository;
using School.UnitOfWork;
using SchoolApp.Models;

SchoolContext context = new SchoolContext();

// création repositories
IUnitOfWorkSchool unitOfWorkSchool = new UnitOfWorkSchoolSQLServer(context);
IRepository<Section> repoSect = unitOfWorkSchool.SectionsRepository;
IRepository<Student> repoStud = unitOfWorkSchool.StudentsRepository;


// ajout de 2 sections
Section sectInfo = new Section { Name = "Info" };
repoSect.Save(sectInfo, s => s.Name.Equals(sectInfo.Name));
Section sectDiet = new Section { Name = "Diet" };
repoSect.Save(sectDiet, s => s.Name.Equals(sectDiet.Name));

// renvoyer toutes les sections
IList<Section> sections = repoSect.GetAll().ToList();

Console.WriteLine("---------- SECTIONS ----------");
foreach (Section s in sections)
{
    Console.WriteLine(s.Name);
}
Console.WriteLine("------------------------------------");
```

2 projet dans une même solution :

1. Télécharger sur MooVin la solution LINQ_DataContext.
2. Ouvrer cette solution en double-cliquant sur le fichier .sln.
3. Ajouter un projet de type Application Console
    a. Clic droit sur la solution -> Ajouter -> Projet
4. Ajouter une référence vers le projet LINQ_DataContext
    a. Dépendances -> clic droit -> ajouter référence au projet LINQ_DataContext
5. Définissez le projet que vous avez créé comme projet de démarrage

a. Clic droit sur le projet -> définir en tant que projet de démarrage

Requête HTTP :

Si on utilise { country } dans l'URL, le nom du paramètre dans l'URL (ici int country) doit correspondre au nom de l'argument dans la méthode.

https://localhost:7128/bank/tva/100/BE

```
app.MapGet("/bank/tva/{price}/{country}", (int price, string country) =>
{
    if (country == "BE")
    {
        return Results.Ok("Prix calculé : " + price * 1.21);
    }
    if (country == "FR")
    {
        return Results.Ok("Prix calculé : " + price * 1.20);
    }
    return Results.BadRequest("Code de pays non valide. Utilisez 'BE' ou 'FR'."); ;
})
.WithName("CalculateTva")
.WithOpenApi();

app.Run()
```

https://localhost:7128/bank/tva?price=100&country=BE

```
app.MapGet("/bank/tva", (int price, string country) =>
{
    if (country == "BE")
    {
        return Results.Ok("Prix calculé : " + price * 1.21);
    }
    if (country == "FR")
    {
        return Results.Ok("Prix calculé : " + price * 1.20);
    }
    return Results.BadRequest("Code de pays non valide. Utilisez 'BE' ou 'FR'."); ;
})
.WithName("CalculateTva")
.WithOpenApi();
```

```
app.Run();
```

Création d'API minimaliste :

1. Créer un projet **ASP.NET Core Web API.**
2. **Décocher la case "use controllers"**
3. Ajouter les routes directement dans program.cs

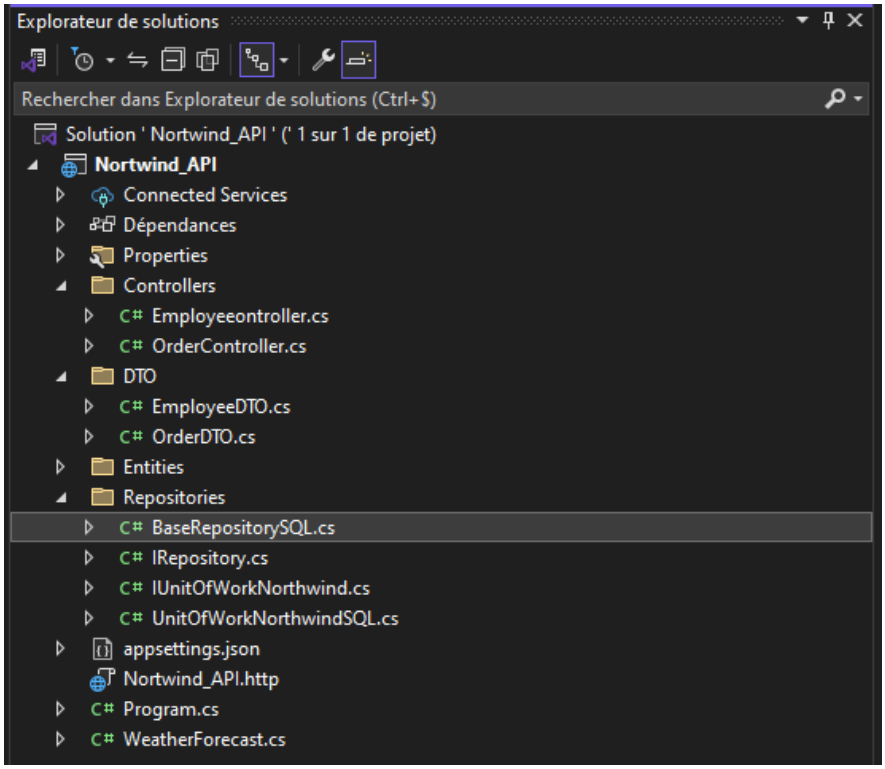
Création d'API:

https://github.com/olivierchoquet/net_solutions/tree/main/Semaine%207%20-%20Recap/Northwind_API

1. Créer un projet **ASP.NET Core Web API**
2. **Cocher la case "use controllers"**
3. Créer un dossier « **Entities** » qui contiendra les modèles générés avec scaffold. (ne pas oublier de créer la db, package nugget et puis faire cette commande scaffold).
4. Créer un dossier «**DTO**» qui contient des modèles DTO car il ne faut pas utiliser directement les Entities comme modèle de transfert dans les controllers.
5. Dans le dossier « **Controllers** » ajouter tous les contrôler.
6. Créer le dossier **Repositories** avec les repo et unit of work.
7. Dans le fichier programs.cs activer le **logging HTTP ! Dans le program.cs, voir github**
8. Testser via swagger, la page s'ouvre automatiquement.

Program.cs est le point d'entrée de l'application.

.NET



Controller :

```
namespace Student_API_Controllers.Controllers
{
    [ApiController] // = à @RestController en Spring
    [Route("/api/[controller]")] // = à @RequestMapping("/api/student") en Spring
    public class StudentController : ControllerBase
    {
        // GET: api/student
        [HttpGet] //c'est éuivalent à GetMapping en Spring
        public IEnumerable<Student> GetAllStudents()
        {
            return _students;
        }
    }
}
```

Attention au nom StudentController doit toujours être NomsAvecSController

Utiliser ces méthodes dans les controller :

```
    private static EmployeeDTO EmployeeToDTO(Employee emp) =>
        new EmployeeDTO
        {
```

```
            EmployeeId = emp.EmployeeId,
            LastName = emp.LastName,
            FirstName = emp.FirstName,
            BirthDate = emp.BirthDate,
            HireDate = emp.HireDate,
            Title = emp.Title,
            TitleOfCourtesy = emp.TitleOfCourtesy

        };

    private static Employee DTOToEmployee(EmployeeDTO emp) =>
        new Employee
        {
            EmployeeId = emp.EmployeeId,
            LastName = emp.LastName,
            FirstName = emp.FirstName,
            BirthDate = emp.BirthDate,
            HireDate = emp.HireDate,
            Title = emp.Title,
            TitleOfCourtesy = emp.TitleOfCourtesy
        };
```

Création d'une base de donnée :

1. . Créer la base de données Northwind à partir du fichier « Northwind4SqlServer.sql»
    a. 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. **Affichage -> Explorateur d'objets SQL SERVER**
    b. **Vous devez vous connecter à une instance SQL Server. Nous utiliserons toujours l'instance : (localdb)\MSSQLLocaldb**

3. Ajouter les package NuGet nécessaires pour EntityFramework :
    a. Clic droit sur le **PROJET** -> Gérer les packages NuGet
        i. Microsoft.EntityFrameworkCore
        ii. Microsoft.EntityFrameworkCore.Design
        iii. Microsoft.EntityFrameworkCore.Tools
        iv. Microsoft.EntityFrameworkCore.SqlServer
        v. Microsoft.EntityFrameworkCore.Proxies

4. Généré les entities (ou Models pour le front mais alors changer la commande) :

Outils -> Gestionnaire du Package Nugget -> Console du gestionnaire
Scaffold-DbContext -OutputDir Entities 'Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Northwind' Microsoft.EntityFrameworkCore.SqlServer

Attention si 2 projets dans un solution, pour le 2eme projet :
Scaffold-DbContext 'Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=School' Microsoft.EntityFrameworkCore.SqlServer -OutputDir Entities -Project "back" -StartupProject "back"

5. Lazy Loading : Ajouter UseLazyLoadingProxies() dans le context créé.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        => optionsBuilder.UseSqlServer("Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Northwind").UseLazyLoadingProxies();
```

.NET

WPF (FRONT):

https://github.com/olivierchoquet/net_solutions/tree/main/Semaine%2010%20-%20WPF%20MVVM%202/WpfEmployee

1. Créez un nouveau projet : **Application WPF**

2. Créer les dossiers :

➔ **Views** : qui va contenir les fichiers MainWindow.xaml et MainWindow.xaml.cs
Attention, il va falloir modifier App.xml :

```
<Application x:Class="examjanvier2023.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:examjanvier2023"
             StartupUri="Views/MainWindow.xaml">
    <Application.Resources>
    </Application.Resources>
</Application>
```

➔ **ViewModels** : EmployeeModel et EmployeeVM
Compléter ces classes en s'aidant de Github.

Dans le fichier MainWindow.xaml.cs ajouter cette ligne:

```
namespace examjanvier2023
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new ProductVM();
        }
    }
}
```

Dans la partie **Window ressource**, définir :

Les templates :

```
<Window.Resources>
    <DataTemplate x:Key="listTemplate">
    <StackPanel Margin="0 5 0 5">
        <Label Content="{Binding ProductId}" HorizontalAlignment="Left" VerticalAlignment="Center"/>
        <Label Content="{Binding ProductName}" HorizontalAlignment="Right" VerticalAlignment="Center"/>
    </StackPanel>
</DataTemplate>
</Window.Resources>
```

Pour utiliser les templates :

```
<Grid>
    <ComboBox Name="cbProducts" ItemsSource="{Binding ProductsList}" ItemTemplate="{StaticResource listTemplate}"  SelectedItem="{Binding SelectedProduct}" HorizontalAlignment="Left" Margin="37,64,0,0" VerticalAlignment="Top" Width="120"/>
</Grid>
```

Les styles :

```
<Window.Resources>
    <Style x:Key="TitreStyle">
        <Setter Property="Label.FontSize" Value="18" />
        <Setter Property="Label.BorderBrush" Value="Black"/>
        <Setter Property="Label.BorderThickness" Value="2"/>
    </Style>
</Window.Resources>
```

Pour utiliser les styles :

```
<Label Content="Top Places" HorizontalAlignment="Left" Margin="145,0,0,0" VerticalAlignment="Top" Width="112" Style="{StaticResource TitreStyle}" Grid.Column="1" Grid.ColumnSpan="2"/>
```

Pour afficher l'élément séléctionné :

```
<ComboBox Grid.Row="0" Grid.ColumnSpan="2"  Name="cbProducts" ItemsSource="{Binding ProductsList}"
ItemTemplate="{StaticResource listTemplate}"  SelectedItem="{Binding SelectedProduct}"
HorizontalAlignment="Left" Margin="30,43,0,0" VerticalAlignment="Top" Width="170" Height="46"
Grid.RowSpan="2"/>
```

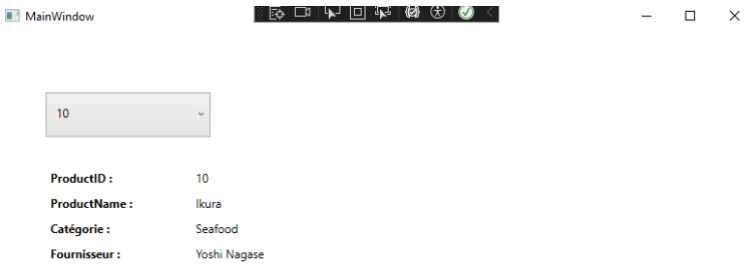
```
<Grid Grid.Row="1" Grid.ColumnSpan="2" DataContext="{Binding SelectedItem, ElementName=cbProducts}"
Margin="30,118,-30,-118">
```

Dans votre application, vous utilisez des **TextBox** liées à des propriétés de votre modèle. Les TextBox, par défaut, utilisent le mode de binding **TwoWay**, car elles permettent de modifier les données. Si la propriété liée n'a pas de setter, la modification n'a aucun effet, ce qui cause l'erreur.

Si on utilise des textbox il faut get et set sur la prop ou alors préciser Mode = OneWay

.NET

Affichage :




```
<Window x:Class="examjanvier2023.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:examjanvier2023"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Window.Resources>
        <DataTemplate x:Key="listTemplate">
            <StackPanel Margin="0 5 0 5">
                <Label Content="{Binding ProductId}" HorizontalAlignment="Left" VerticalAlignment="Center"/>
                <Label Content="{Binding ProductName}" HorizontalAlignment="Right" VerticalAlignment="Center"/>
            </StackPanel>
        </DataTemplate>
    </Window.Resources>

    <Grid Margin="20">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="150"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>

        <!-- ComboBox -->
```

```
        <ComboBox Grid.Row="0" Grid.ColumnSpan="2"  Name="cbProducts" ItemsSource="{Binding
ProductsList}" ItemTemplate="{StaticResource listTemplate}"  SelectedItem="{Binding SelectedProduct}"
HorizontalAlignment="Left" Margin="30,43,0,0" VerticalAlignment="Top" Width="170" Height="46"
Grid.RowSpan="2"/>

        <Grid Grid.Row="1" Grid.ColumnSpan="2" DataContext="{Binding SelectedItem,
ElementName=cbProducts}" Margin="30,118,-30,-118">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="150"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>

            <!-- Labels -->
            <Label Content="ProductID :" FontWeight="Bold" Grid.Row="0" Grid.Column="0"
VerticalAlignment="Center"/>
            <Label Content="ProductName :" FontWeight="Bold" Grid.Row="1" Grid.Column="0"
VerticalAlignment="Center"/>
            <Label Content="Catégorie :" FontWeight="Bold" Grid.Row="2" Grid.Column="0"
VerticalAlignment="Center"/>
            <Label Content="Fournisseur :" FontWeight="Bold" Grid.Row="3" Grid.Column="0"
VerticalAlignment="Center"/>

            <!-- Values -->
            <TextBlock Text="{Binding ProductId}" Grid.Row="0" Grid.Column="1" VerticalAlignment="Center"
Margin="5,0,0,0"/>
            <TextBlock Text="{Binding ProductName}" Grid.Row="1" Grid.Column="1" VerticalAlignment="Center"
Margin="5,0,0,0"/>
            <TextBlock Text="{Binding CategoryName}" Grid.Row="2" Grid.Column="1" VerticalAlignment="Center"
Margin="5,0,0,0"/>
            <TextBlock Text="{Binding Supplier}" Grid.Row="3" Grid.Column="1" VerticalAlignment="Center"
Margin="5,0,0,0"/>
        </Grid>
    </Grid>
</Window>
```

Est-ce que vous voulez que quan don ecrit ça s'affiche directement en temps réel ou bien ça se change quand je quitte l'input ? si en temps réel alors mettre UpdatTRigger

DelegateCommand :

```
<Button  Command ="{Binding AbandonateCommand}" Content="Abandonné le produit sélectionné" Width="249"
Margin="0,314,471,26" HorizontalAlignment="Right" Grid.Row="1" Grid.ColumnSpan="2"/>
```

```
using examjanvier2023.Models;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using WpfApplication1.ViewModels;

namespace examjanvier2023.ViewModels
{
   public class ProductVM
   {
      private NorthwindContext dc = new NorthwindContext();
      private ProductModel _selectedProduct;
      private ObservableCollection<ProductModel> _ProductsList;
      private DelegateCommand _abandonnateCommand;


      public DelegateCommand AbandonateCommand
      {
         get { return _abandonnateCommand = _abandonnateCommand ?? new DelegateCommand(AbandonateProduct); }
      }
```

```
    private void AbandonateProduct()
    {
        if (SelectedProduct == null)
        {
            MessageBox.Show("Aucun produit sélectionné !");
            return;
        }
        Product verif = dc.Products.Where(e => e.ProductId == SelectedProduct.MonProduct.ProductId).SingleOrDefault();
        if (verif != null)
        {
            verif.Discontinued = true;
            dc.SaveChanges();
            ProductsList.Remove(SelectedProduct);
        }

        MessageBox.Show("Enregistrement en base de données fait");

    }

}
```

Attention, si on est définit dans par exemple un grid pour qui on avait changé le context, tout ce qui est dans le grid hérite de ce context. Donc si on veut un autre context :

```
  <Grid Grid.Row="1" Grid.ColumnSpan="2" DataContext="{Binding SelectedItem, ElementName=cbProducts}"
Margin="30,118,-30,-118">

    <!-- Labels -->
    <Label Content="ProductID :" FontWeight="Bold" Grid.Row="0" Grid.Column="0"
VerticalAlignment="Center"/>
    <Label Content="ProductName :" FontWeight="Bold" Grid.Row="1" Grid.Column="0"
VerticalAlignment="Center"/>
    <Label Content="Fournisseur :" FontWeight="Bold" Grid.Row="2" Grid.Column="0"
VerticalAlignment="Center"/>
    <Label Content="Quantité par unit :" FontWeight="Bold" Grid.Row="3" Grid.Column="0"
VerticalAlignment="Center"/>

    <!-- Values -->
    <TextBox  Text="{Binding ProductId}" Grid.Column="1" VerticalAlignment="Center" Margin="5,0,448,0"
IsReadOnly="True"/>
```

```
      <TextBox  Text="{Binding ProductName, UpdateSourceTrigger=PropertyChanged}" Grid.Row="1"
Grid.Column="1" VerticalAlignment="Center" Margin="5,0,448,0"/>
      <TextBox  Text="{Binding Supplier}" Grid.Row="2" Grid.Column="1" VerticalAlignment="Center"
Margin="5,0,448,0" IsReadOnly="True"/>
      <TextBox  Text="{Binding QuantityPerUnit, UpdateSourceTrigger=PropertyChanged}" Grid.Row="3"
Grid.Column="1" VerticalAlignment="Center" Margin="5,0,448,0"/>
      <Button  Content="Maj" Grid.Row="4" Grid.Column="1" Margin="5,10,0,0" Width="80"
            Command="{Binding DataContext.SaveCommand, RelativeSource={RelativeSource FindAncestor,
AncestorType={x:Type Window}}}" HorizontalAlignment="Left"/>
   </Grid>
</Grid>
```

ObservableCollection :

Si le produit disparaît directement de la liste dans la ComboBox après avoir été marqué comme "discontinued", c'est parce que vous utilisez une **ObservableCollection** pour ProductsList. Voici pourquoi cela fonctionne sans avoir explicitement appelé OnPropertyChanged :

Pourquoi ça marche sans OnPropertyChanged :

1. **ObservableCollection gère automatiquement les notifications :**
   - Lorsque vous appelez ProductsList.Remove(SelectedProduct), la ObservableCollection déclenche automatiquement une notification de changement.
   - Cela met à jour l'interface utilisateur (ComboBox) liée à ProductsList.
2. **Pas besoin de notifier manuellement l'ensemble de la liste :**
   - OnPropertyChanged(nameof(ProductsList)) n'est nécessaire que si vous affectez une nouvelle instance à ProductsList.
   - Dans votre cas, vous modifiez directement le contenu de la liste existante, donc la notification est gérée par ObservableCollection.

Quand utiliser OnPropertyChanged :

1. **Si vous changez toute la collection :**
   - Exemple : Si vous remplacez l'intégralité de ProductsList par une nouvelle collection.

ObservableCollection implémente déjà INotifyCollectionChanged, ce qui signifie qu'elle notifie automatiquement l'interface quand on :

- Ajoute un élément
- Supprime un élément
- Remplace un élément
- Efface la collection

Exam 2023 erreur :

Attention à ne pas oublier de mettre les getter setter en public.
La cause de l'erreur est que la propriété CategoryName dans la classe Category est définie avec un accesseur get et un accesseur set, ce qui signifie qu'elle est en lecture-écriture. Cependant, l'erreur indique que la propriété CategoryName de la classe ProductModel est en lecture seule. Pour résoudre ce problème, il est important de vérifier la définition de la classe ProductModel et de s'assurer que la propriété CategoryName est également en lecture-écriture Si la propriété CategoryName dans ProductModel est en lecture seule, vous devez la rendre en lecture-écriture pour permettre une liaison TwoWay ou OneWayToSource.

```
using Exam2023.Models;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

namespace Exam2023.ViewModels
{
    public class ProductVM : INotifyPropertyChanged
    {
        private NorthwindContext dc = new NorthwindContext();



        public event PropertyChangedEventHandler PropertyChanged; // La view s'enregistera automatiquement sur
cet event
        protected virtual void OnPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName)); // On notifie que la propriété a
changé
            }
        }

        private ProductModel _selectedProduct;
        private ObservableCollection<ProductModel> _ProductsList;
```

```
private ObservableCollection<ProductModel> _ProductsByCountry;

private DelegateCommand _abandonnateProduct;

public ObservableCollection<ProductModel> ProductsList
{
    get
    {
        if (_ProductsList == null)
        {
            _ProductsList = loadProduct();
        }

        return _ProductsList;

    }

}


private ObservableCollection<ProductModel> loadProduct()
{
    ObservableCollection<ProductModel> localCollection = new ObservableCollection<ProductModel>();
    foreach (var item in dc.Products)
    {
        if(item.Discontinued == true)
        {
            continue;
        }
        localCollection.Add(new ProductModel(item));

    }

    return localCollection;

}

public ProductModel SelectedProduct
{
    get { return _selectedProduct; }
```

```
        set { _selectedProduct = value; OnPropertyChanged(nameof(SelectedProduct)); }
    }



    public DelegateCommand AbandonProductCommand
    {
        get { return _abandonnateProduct = _abandonnateProduct ?? new
DelegateCommand(AbandonnateProduct); }
    }


    private void AbandonnateProduct()
    {
        if (SelectedProduct?.MonProduct == null)
        {
            MessageBox.Show("Selected product is not set.");
            return;
        }
        Product productToUpdate = dc.Products.Where(e => e.ProductId ==
SelectedProduct.MonProduct.ProductId).SingleOrDefault();
        if (productToUpdate != null)
        {
            productToUpdate.Discontinued = true;
            dc.SaveChanges();
            _ProductsList.Remove(SelectedProduct);
        }
        MessageBox.Show("Enregistrement en base de données fait");
    }

  }
}
```

```
<Window x:Class="Exam2023.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:Exam2023"
```

```
      mc:Ignorable="d"
      Title="MainWindow" Height="450" Width="800">

   <Window.Resources>
      <DataTemplate x:Key="listTemplate">
         <StackPanel Margin="0 5 0 5">
            <Label Content="{Binding ProductId}" HorizontalAlignment="Left" VerticalAlignment="Center"/>
            <Label Content="{Binding ProductName}" HorizontalAlignment="Right" VerticalAlignment="Center"/>
         </StackPanel>
      </DataTemplate>
   </Window.Resources>


   <Grid>
      <Grid.RowDefinitions>
         <RowDefinition Height="22*"/>
         <RowDefinition Height="9*"/>
      </Grid.RowDefinitions>

      <ComboBox Name="ProductsComboBox"
        ItemsSource="{Binding ProductsList}" Margin="27,39,435,177"  SelectedItem="{Binding SelectedProduct}"
ItemTemplate="{StaticResource listTemplate}">
      </ComboBox>


      <Grid Margin="27,154,419,38" DataContext="{Binding SelectedItem, ElementName=ProductsComboBox}"
Grid.RowSpan="2" >
         <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100"/>
            <ColumnDefinition Width="*"/>
         </Grid.ColumnDefinitions>
         <StackPanel>
            <Label Content="Product Id :" Margin="6"  />
            <Label Content="Product Name :" Margin="6" />
            <Label Content="Category :" Margin="6" />
            <Label Content="Fournisseur :" Margin="6" />
         </StackPanel>

         <StackPanel Grid.Column="1">
            <TextBox Height="27" Margin="6,6,6,5" Text="{Binding ProductId}" />
```

```
            <TextBox Height="27" Margin="6,6,6,5" Text="{Binding ProductName}"/>
            <TextBox Height="27" Margin="6,6,6,5" Text="{Binding CategoryName}"/>
            <TextBox Height="27" Margin="6,6,6,5" Text="{Binding SupplierName}"/>
        </StackPanel>
    </Grid>

    <Button Command ="{Binding AbandonProductCommand}" Content="Abandonner le produit sélectionné"
Width="174" Margin="92,53,0,38" Grid.Row="1" HorizontalAlignment="Left"/>


  </Grid>
</Window>
```

```
using Exam2023.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exam2023.ViewModels
{
    public class ProductModel : INotifyPropertyChanged
    {
        private readonly Product _monProduct;

        public Product MonProduct
        {
            get { return _monProduct; }
        }


        // Property changed standard handling
        public event PropertyChangedEventHandler PropertyChanged; // La view s'enregistera automatiquement sur
cet event
        protected virtual void OnPropertyChanged(string propertyName)
        {
```

```
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName)); // On notifie que la propriété a
changé
        }
    }

    public ProductModel(Product current)
    {
        this._monProduct = current;
    }


    public String ProductName
    {
        get { return _monProduct.ProductName; }
        set
        {
            _monProduct.ProductName = value;
            //OnPropertyChanged("ProductName");
        }
    }

    public int ProductId
    {
        get { return _monProduct.ProductId; }
        set
        {
            _monProduct.ProductId = value;
        }
    }

    public string CategoryName
    {
        get { return _monProduct.Category?.CategoryName ?? "No Category"; }
        set
        {
            if (_monProduct.Category != null)
            {
                _monProduct.Category.CategoryName = value;
```

```
            }
        }
    }

    public string SupplierName
    {
        get { return _monProduct.Supplier?.ContactName ?? "No Supplier"; }
        set
        {
            if (_monProduct.Supplier != null)
            {
                _monProduct.Supplier.ContactName = value;
            }
        }
    }
  }
}
```

```
using Exam2023.Models;
using System.Text;
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 Exam2023.ViewModels;

namespace Exam2023
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
```

```
  {
      public MainWindow()
      {
          InitializeComponent();
          this.DataContext = new ProductVM();
      }
  }
}
```

Conventions et équivalence en java:

En C#, le mot-clé **virtual** permet de déclarer qu'une méthode ou une propriété peut être **remplacée (overridden)** dans une classe dérivée.

```
public virtual void Display() => Console.WriteLine("Base Class");


public override void Display() => Console.WriteLine("Derived Class");
```

Les noms de méthodes et des propriétés doivent commencer par une majuscule et utiliser le **PascalCase**.

Les noms de variable privé doivent commencer par une _ et minuscule et utiliser le **camelCase**.

```
private string _firstName; // Champ privé


public string FirstName // Propriété publique avec PascalCase
{
    get => _firstName;
    set => _firstName = value;
}
```

Lorsqu'une classe est marquée avec le modificateur internal, elle est **accessible uniquement à partir du même assemblage** (du même projet ou bibliothèque).

Implémente Serializable devient :

```
[Serializable]
public class Actor
```

Extends Person deviant :

```
public class Actor : Person
```

Un attribute final ➔ readonly

```
super(name, firstname, birthDate); devient
// devient :
```

```
public Actor(string name, string firstname, DateTime birthDate, int sizeInCentimeter)
    : base(name, firstname, birthDate)
```

L'équivalent du package Java est le **namespace. Attention si je veux utiliser ma classe autre part (avec using), il faut mettre le namespace !**

Si on veut utiliser une classe définie dans un autre fichier :

- ⇨ Placer la classe dans un namespace.
- ⇨ Importer ce namespace avec using dans le fichier où vous voulez utiliser la classe.

Pour utiliser par exemple dans program.cs des classes du dossier models :

```
//Fichier 1 :
namespace MyApp.Models

//Fichier 2 :
using MyApp.Models;
```

Attention, namespace ne veut pas d'office dire même dossier !

```
//Fichier 1 :
namespace StringExtensions;
public static class StringExtensions

//Fichier 2 :
using StringExtensions;
Console.WriteLine("test IsAPalindrome");
```

Deleguate :

Un **délégué** est une sorte de pointeur de méthode. C'est une référence à une méthode qui peut être passée comme un paramètre ou exécutée dynamiquement. Cela permet de définir une méthode sans savoir à l'avance laquelle sera appelée, ce qui rend le code très flexible.

- ⇨ Un délégué représente une ou plusieurs méthodes ayant une signature spécifique (même type de retour et paramètres).
- ⇨ On peut l'utiliser pour appeler toutes les méthodes qu'il référence.
- ⇨ Multicast Delegates : Ajouter plusieurs méthodes à un délégué.

```
// Déclaration du délégué
public delegate void DelegateLog(string msg);

private DelegateLog _delegateCallBack;
public DelegateLog DelegateCallBack
{
get { return _delegateCallBack; }
set { _delegateCallBack = value; }
}

public void LogMessage(string message)
{
_delegateCallBack?.Invoke(message);
}


// Méthodes qui correspondent à la signature du délégué
public static void LogToConsole(string msg)
{
Console.WriteLine(msg);
}


public static void LogToFile(string msg)
{
    string docPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

    using (StreamWriter outputFile = new StreamWriter(Path.Combine(docPath, "log.txt")))
    {
        outputFile.WriteLine(msg);
    }
```

```
}


// Utilisation du délégué
    public static void Main(string[] args)
    {
        Logger log = new Logger();
        log.DelegateCallBack += ConsoleLogger.LogToConsole;
        log.DelegateCallBack += FileLogger.LogToFile;

        log.LogMessage("test");
    }
```