

## Chapitre 4. Développement d'interfaces graphiques et Accès aux bases de données

### 4.1. Windows Presentation Foundation

Windows Presentation Foundation (WPF) est un framework d'interface utilisateur (IU) permettant de créer des applications clientes de bureau.

La plateforme de développement WPF prend en charge un large éventail de fonctionnalités de développement d'applications, notamment un modèle d'application, des ressources, des contrôles, des graphiques, une mise en page, une liaison de données, des documents et la sécurité.

WPF fait partie de .NET et utilise le langage XAML (Extensible Application Markup Language) pour fournir un modèle déclaratif de programmation d'applications.

Nous allons apprendre à créer une application desktop moderne avec WPF en suivant le pattern MVVM

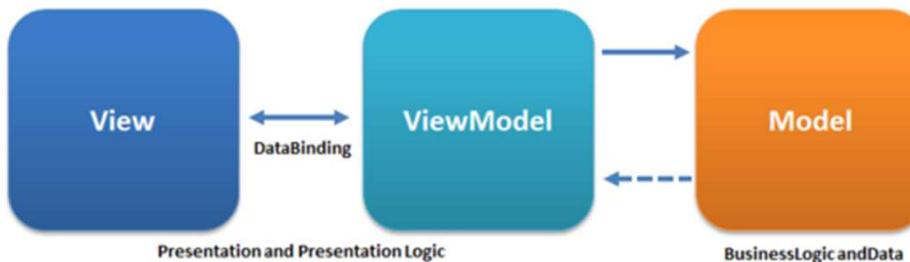


Ass. Eric Wangi  
243812961556

## 4.2. Modèle-vue-vue modèle

Le **modèle-vue-vue modèle** (en abrégé **MVVM**, de l'[anglais](#) *Model View ViewModel*) est une architecture et une méthode de conception utilisée dans le génie logiciel

Apparu en 2004<sup>[3]</sup>, MVVM est originaire de [Microsoft](#) et adapté pour le développement des applications basées sur les technologies [Windows Presentation Foundation](#) et [Silverlight](#)<sup>[4]</sup> via l'outil MVVM Light<sup>[5]</sup> par exemple. Cette méthode permet, tel le modèle MVC ([modèle-vue-contrôleur](#)), de séparer la vue de la logique et de l'accès aux données en accentuant les principes de [liaison](#) et d'[événement](#).



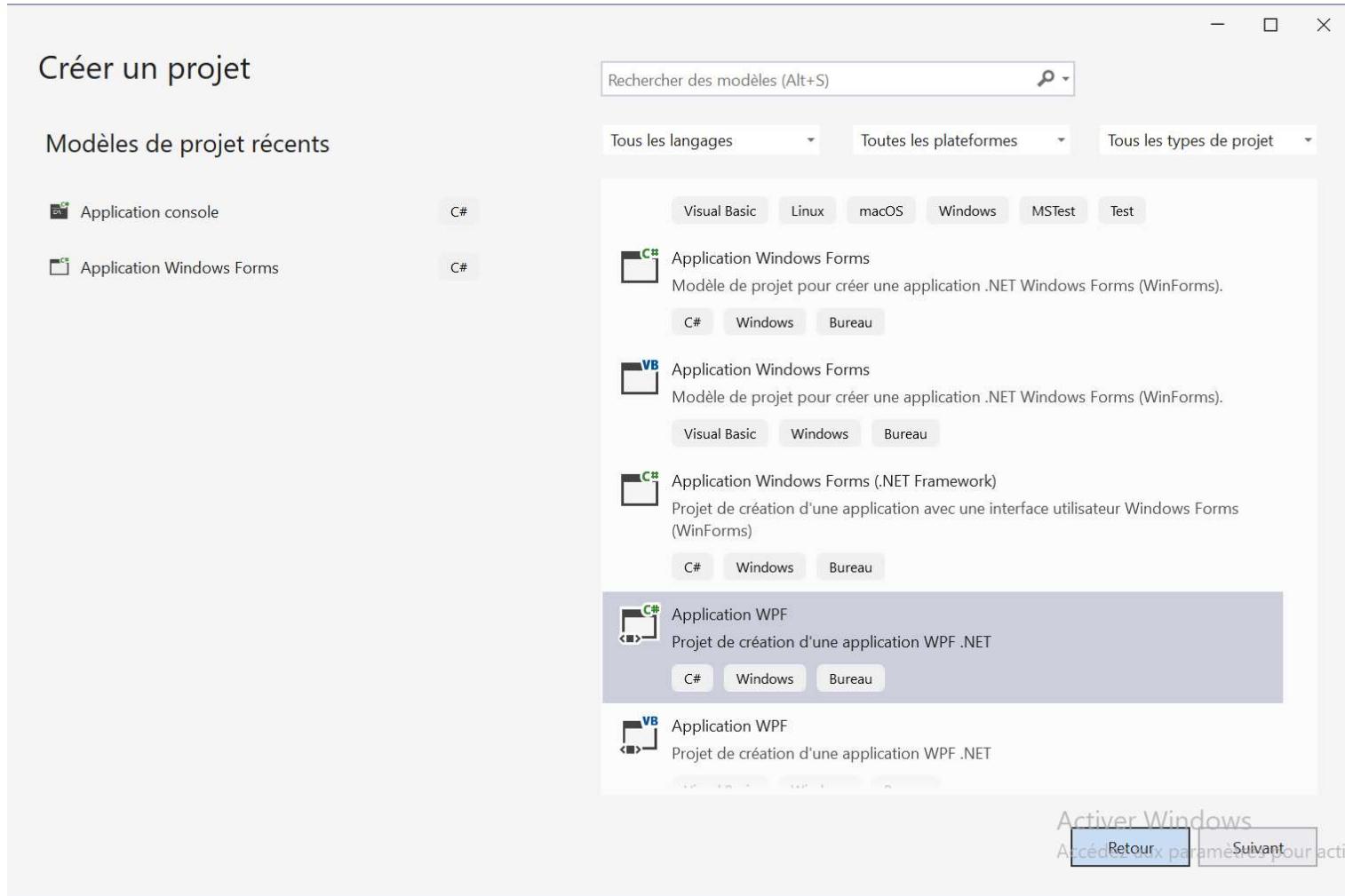
Ass. Eric Wangi  
243812961556

## 4.2. Création du projet WPF

A partir du Visual Studio, procédez de la manière suivante:



Ass. Eric Wangi  
243812961556



Ass. Eric Wangi  
243812961556

## Nommons le projet:

Configurer votre nouveau projet

Application WPF    C#    Windows    Bureau

Nom du projet

Emplacement

 ...

Nom de la solution ⓘ

Placer la solution et le projet dans le même répertoire

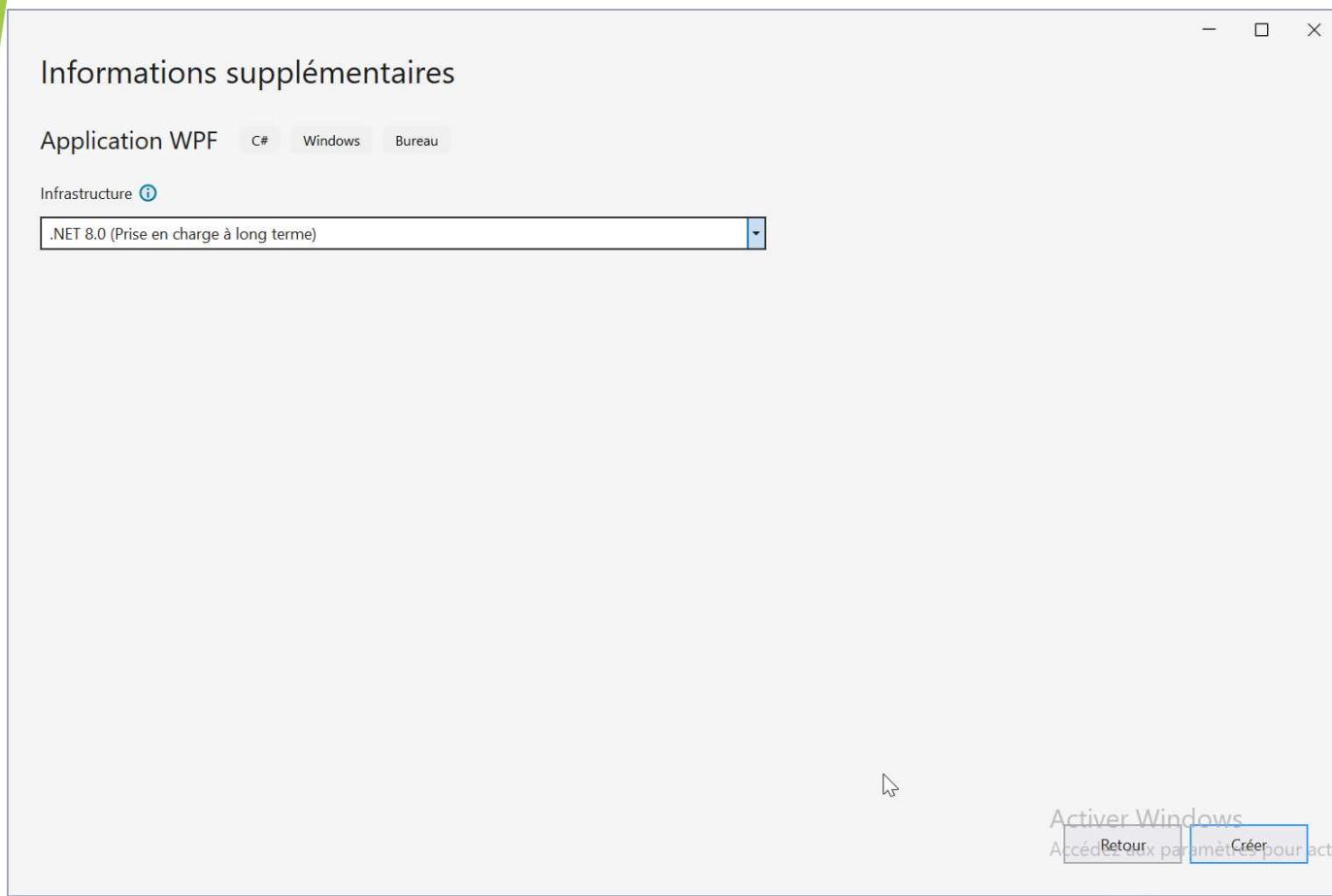
Projet sera créé en tant que « C:\Users\ERIWANG\Documents\projetcsharp\EasyApp\ »

Activer Windows  
Accédez aux paramètres pour activer

Retour    Suivant



Ass. Eric Wangi  
243812961556

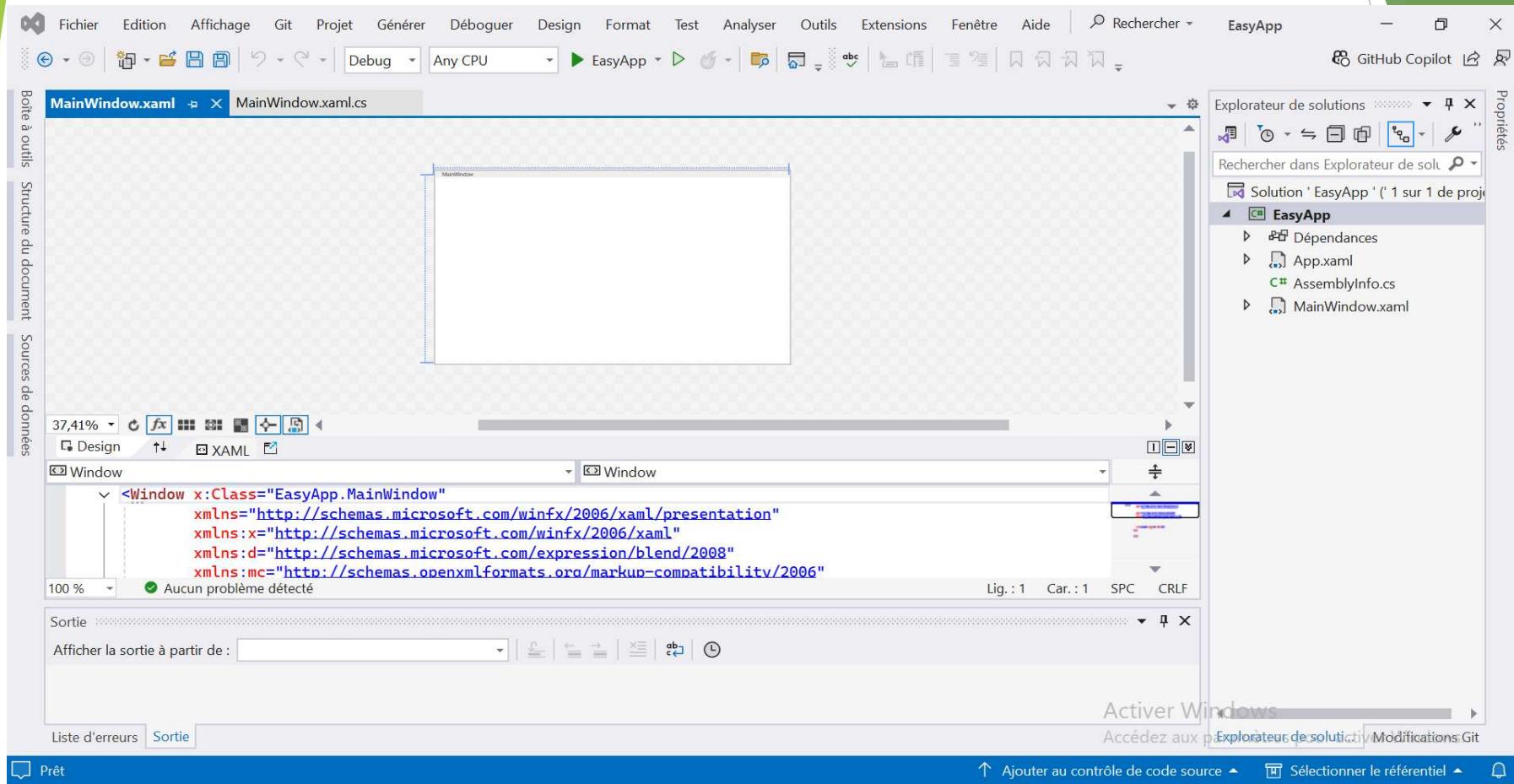


Cliquez sur créer.

Ass. Eric Wangi  
243812961556



Le projet vient d'être créé:



Ass. Eric Wangi  
243812961556

MainWindow.xaml MainWindow.xaml.cs

Design XAML

Window

```
<Window x:Class="EasyApp.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:EasyApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
    </Grid>
</Window>
```

Explorateur de solutions

Solution 'EasyApp' (1 sur 1 de projet)

- EasyApp
  - Dépendances
  - App.xaml
  - C# AssemblyInfo.cs
  - MainWindow.xaml

Propriétés

Rechercher dans Explorateur de sol...

100 % Aucun problème détecté



Ass. Eric Wangi  
243812961556

### 4.3. Création du formulaire

Nous allons créer notre première page.

Au niveau de la balise **Grid**, nous avons la possibilité de définir nos lignes et nos colonnes.  
Dans ce projet, nous aurons besoin des lignes.

Cette définition est rendue possible grâce à la balise :

```
<Grid.RowDefinitions>
  |
</Grid.RowDefinitions>
```

Nous allons prendre la taille de notre fenêtre selon la réaliéde champs qui Seront placés. Raison pour laquelle l'hauteur sera Auto.

Le premier élément que nous allons placé est le **StackPanel** qui est juste un Panel que nous l'avons positionné à la ligne 0. Pour le StackPanel place, nous devons spécifié l'orientation qui peut être vertical ou horizontal.



Ass. Eric Wangi  
243812961556

Nous allons mettre notre code dans **MainWindows.xaml.cs** juste après la zone de ressource (Windows.Ressource):

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel Orientation="Vertical" Grid.Row="0">
        </StackPanel>
    </Grid>
```

Plaçons un **TextBlock** pour définir le texte, le **comboBox** pour afficher la liste des facultés et les boutons de commande:



Ass. Eric Wangi  
243812961556

```
<Window x:Class="EasyApp.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:EasyApp"
    mc:Ignorable="d"
    Title="MainWindow" Height="450" Width="800">
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel Orientation="Vertical" Grid.Row="0">
        <TextBlock Text="Nom"/>
        <TextBox />
        <TextBlock Text="Prenom "/>
        <TextBox />
        <TextBlock Text="Faculté"/>

        <ComboBox>
            <ComboBoxItem Content="FASI"/>
            <ComboBoxItem Content="FASE"/>
            <ComboBoxItem Content="DROIT"/>
            <ComboBoxItem Content="MEDECINE"/>
            <ComboBoxItem Content="THEOLOGIE"/>
        </ComboBox>
        <StackPanel Orientation="Horizontal">
            <Button Content="Enregistrer"/>
            <Button Content="Modifier"/>
            <Button Content="Supprimer"/>
        </StackPanel>
    </StackPanel>
</Grid>
</Window>
```



Ass. Eric Wangi  
243812961556

## Exécutions

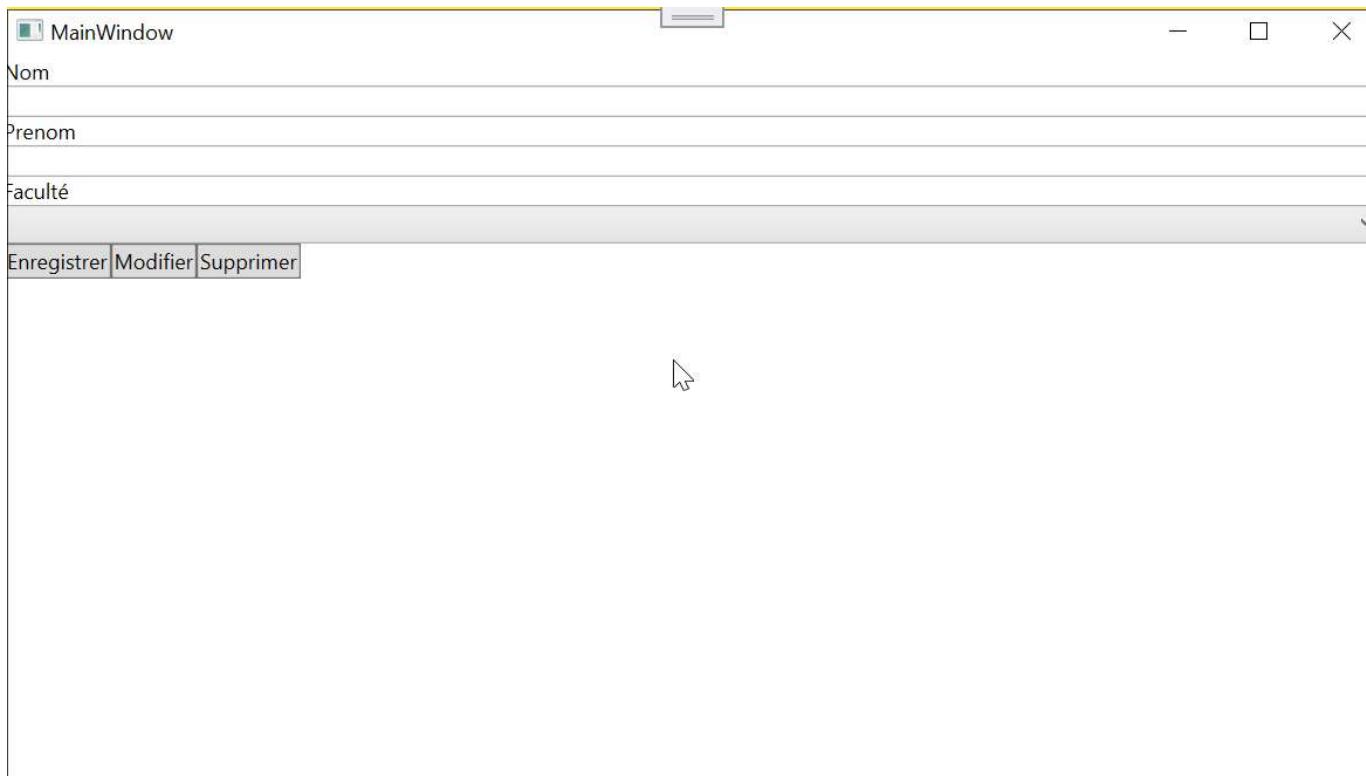
MainWindow

Nom

Prenom

Faculté

Enregistrer Modifier Supprimer



Ass. Eric Wangi  
243812961556

Nous allons maintenant définir un style globale pour chacun de composant placé dans notre formulaire. Pour cela, nous devons nous placer dans Window.Ressource:

## 1. Style pour le TextBox

```
<Style TargetType="TextBox">
    <Setter Property="Padding" Value="5"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="BorderBrush" Value="#Black"/>
    <Setter Property="Background" Value="White"/>
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="Margin" Value="0,0,0,10"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="TextBox">
                <Border CornerRadius="5" Background="{TemplateBinding Background}"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    Padding="{TemplateBinding Padding}"
                    BorderThickness="{TemplateBinding BorderThickness}">
                    <ScrollViewer x:Name="PART_ContentHost"/>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```



Ass. Eric Wangi  
243812961556

## 2. Style pour le comboBox

```
<Style TargetType="ComboBox">
    <Setter Property="Padding" Value="10"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="BorderBrush" Value="Black"/>
    <Setter Property="Background" Value="White"/>
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="Margin" Value="0,0,0,10"/>
</Style>
```



Ass. Eric Wangi  
243812961556

## 2. Style pour le bouton

```
<Style TargetType="Button">
    <Setter Property="Padding" Value="15,8"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="Background" Value="#Green"/>
    <Setter Property="BorderThickness" Value="0"/>
    <Setter Property="Margin" Value="0,0,10,10"/>
    <Setter Property="Foreground" Value="White"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Border CornerRadius="5" Background="{TemplateBinding Background}"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    Padding="{TemplateBinding Padding}"
                    BorderThickness="{TemplateBinding BorderThickness}">
                    <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```



Ass. Eric Wangi  
243812961556

Ajoutons aussi une couleur de background de bouton.

```
<Button Content="Enregistrer" Background="#blue"/>
<Button Content="Modifier" Background="#Green"/>
<Button Content="Supprimer" Background="#Red"/>
```

Le code complet :

```
<Window x:Class="EasyApp.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:EasyApp"
    mc:Ignorable="d"
    Title="MainWindow" Height="450" Width="800">
<Window.Resources>
    <Style TargetType="TextBox">
        <Setter Property="Padding" Value="5"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="BorderBrush" Value="#Black"/>
        <Setter Property="Background" Value="White"/>
        <Setter Property="BorderThickness" Value="1"/>
        <Setter Property="Margin" Value="0,0,0,10"/>
        <Setter Property="Template">
```



Ass. Eric Wangi  
243812961556

```

<Setter.Value>
    <ControlTemplate TargetType="TextBox">
        <Border CornerRadius="5" Background="{TemplateBinding Background}"
            BorderBrush="{TemplateBinding BorderBrush}"
            Padding="{TemplateBinding Padding}"
            BorderThickness="{TemplateBinding BorderThickness}">
            <ScrollViewer x:Name="PART_ContentHost"/>
        </Border>
    </ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style TargetType="ComboBox">
    <Setter Property="Padding" Value="10"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="BorderBrush" Value="Black"/>
    <Setter Property="Background" Value="White"/>
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="Margin" Value="0,0,0,10"/>
</Style>

<Style TargetType="Button">
    <Setter Property="Padding" Value="15,8"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="Background" Value="Green"/>
    <Setter Property="BorderThickness" Value="0"/>
    <Setter Property="Margin" Value="0,0,10,10"/>
    <Setter Property="Foreground" Value="White"/>
    <Setter Property="Template">

```



Ass. Eric Wangi  
243812961556

```

<Setter.Value>
    <ControlTemplate TargetType="Button">
        <Border CornerRadius="5" Background="{TemplateBinding Background}"
            BorderBrush="{TemplateBinding BorderBrush}"
            Padding="{TemplateBinding Padding}"
            BorderThickness="{TemplateBinding BorderThickness}">
            <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
        </Border>
    </ControlTemplate>
</Setter.Value>
</Setter>
</Style>

</Window.Resources>

```

```

<Grid Margin="20">
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!-- formulaire -->
<StackPanel Orientation="Vertical" Grid.Row="0">
    <TextBlock Text="Nom"/>
    <TextBox />
    <TextBlock Text="Prenom "/>
    <TextBox />
    <TextBlock Text="Faculté"/>
    <ComboBox>
        <ComboBoxItem Content="FASI"/>
        <ComboBoxItem Content="FASE"/>
        <ComboBoxItem Content="DROIT"/>
        <ComboBoxItem Content="MEDECINE"/>
        <ComboBoxItem Content="THEOLOGIE"/>
    </ComboBox>
</StackPanel>

```



Ass. Eric Wangi  
243812961556

```
<StackPanel Orientation="Horizontal">
    <Button Content="Enregistrer" Background="#Blue" />
    <Button Content="Modifier" Background="#Green"/>
    <Button Content="Supprimer" Background="#Red" />
</StackPanel>
</StackPanel>

<!-- fin formulaire -->

</Grid>
</Window>
```

## Exécutons

MainWindow



Nom

Prenom

Faculté

Enregistrer   Modifier   Supprimer



Ass. Eric Wangi  
243812961556

#### 4.4.Ajout de la listview

La listView est un composant sous forme de grille permettant d'afficher les données sous forme des lignes et des colonnes.

Nous allons placé notre ListView après le StackPanel placé précédemment.

```
<!-- Affichage d'une grille pour la liste des étudiants -->
<ListView Grid.Row="1">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Nom" Width="250"/>
            <GridViewColumn Header="Prenom" Width="100"/>
            <GridViewColumn Header="Faculté" Width="100"/>
        </GridView>
    </ListView.View>
</ListView>
```



Ass. Eric Wangi  
243812961556

MainWindow

Nom

Prenom

Faculté

Enregistrer   Modifier   Supprimer

Nom	Prenom	Faculté



Ass. Eric Wangi  
243812961556

## 4.5.Le model

Créons un dossier dans notre projets nommé models qui servira à stocker nos différents models.

Ensute, nous allons créer une classe nommé Etudiant qui va représenter la table Etudiant dans la base de données.

Voici la règle de gestion: Tout étudiant appartient à une et une seule promotion.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EasyApp.models
{
    1 référence
    public class Etudiant
    {
        0 références
        public int Id { get; set; }
        0 références
        public string nom { get; set; }
        0 références
        public string prenom { get; set; }
        0 références
        public int faculteId { get; set; }

        0 références
        public Faculte faculte { get; set; }
    }
}
```



Ass. Eric Wangi  
243812961556

Créons aussi la classe Faculte:

Règle de gestion: Dans une faculté appartient un ou plusieurs étudiants.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EasyApp.models
{
    8 références
    public class Faculte
    {
        1 référence
        public int id { get; set; }
        2 références
        public string nomFaculte { get; set; }

        1 référence
        public List<Etudiant> etudiants { get; set; }

    }
}
```



Ass. Eric Wangi  
243812961556

#### 4.6. Accès à la base de données avec EntityFramework core

Pour gérer l'accès et manipuler notre base de données depuis l'application, il est préférable d'utiliser un ORM pouvant nous faciliter cette tâche.

**EntityFramework** est un ORM de Microsoft qui est destiné à faciliter la gestion de la BD sans pour autant écrire une requête SQL. Nous aurons tout simplement Besoin d'avoir des connaissant dans l'utilisation du langage Linq.



Ass. Eric Wangi  
243812961556

#### 4.6.1.Ajout du DBContext

Créons un dossier nommé ApplicationDbContext.

Ajoutons dans ce dossier une classe **AppDbContext**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

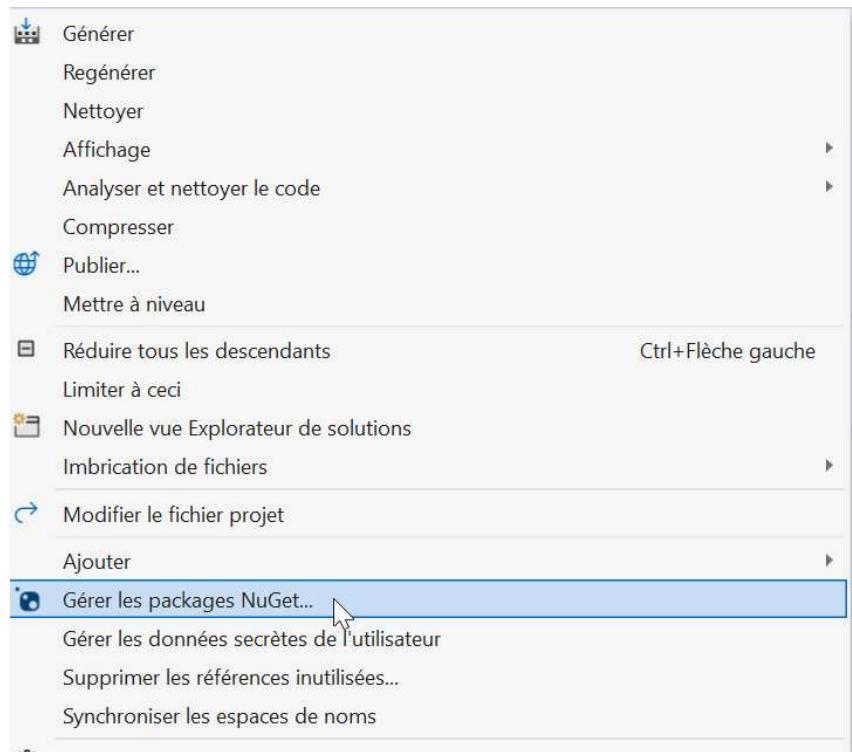
namespace EasyApp.ApplicationDBContexte
{
    0 références
    public class AppDbContext
    {
    }
}
```

Avant d'aller plus loin, nous devons intégré **Entity Framework** dans notre projet:



Ass. Eric Wangi  
243812961556

Clic droit sur le nom de notre projet:



Ass. Eric Wangi  
243812961556

## Tapez EntityFramework tools

The screenshot shows the Visual Studio interface with the NuGet Package Manager open. The search bar at the top contains "Entityframework tools". Below it, the results list includes:

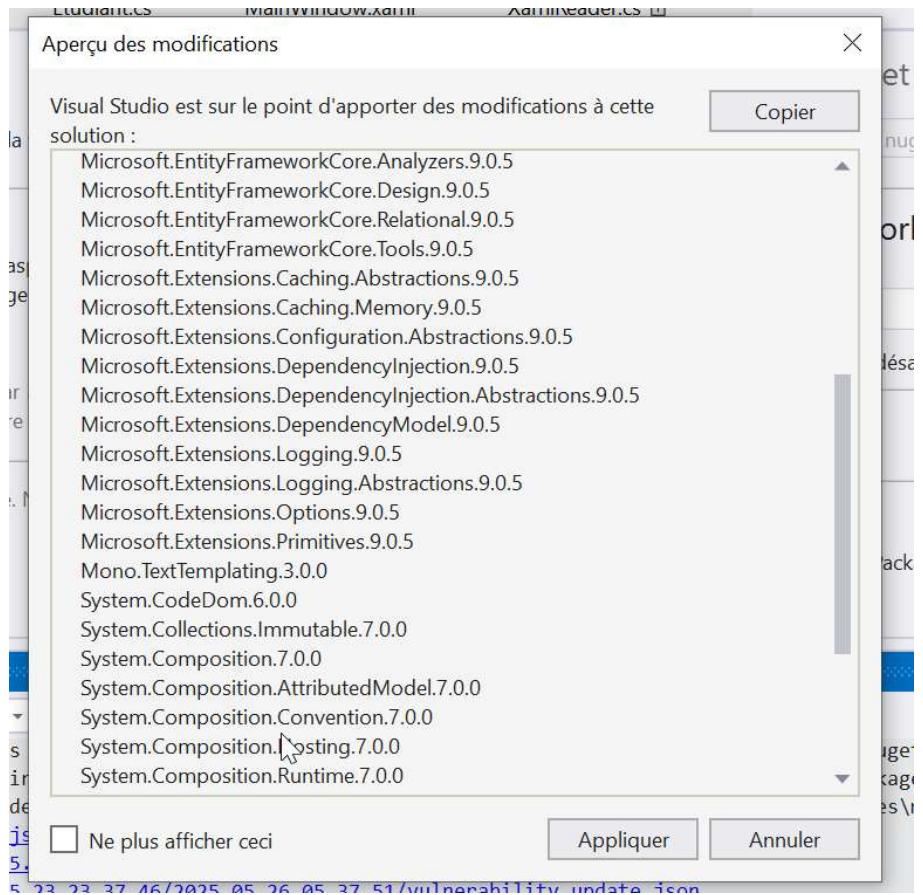
- Microsoft.EntityFrameworkCore.Tools** (par aspnet, dotnetframework, En... 9.0.5) - Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.
- Microsoft.EntityFrameworkCore.Design** (par aspnet, dotnetframework, I... 9.0.5) - Shared design-time components for Entity Framework Core tools.
- dotnet-ef** (par aspnet, dotnetframework, Microsoft, 76,3M téléchargements 9.0.5) - Entity Framework Core Tools for the .NET Command-Line Interface.
- Microsoft.EntityFrameworkCore.Tools.DotNet** (par aspnet, EntityFr... 2.0.3) - Entity Framework Core .NET Command Line Tools. Includes dotnet-ef.

The "Microsoft.EntityFrameworkCore.Tools" item is selected, showing its details page. The "Version" dropdown is set to "Dernière version stable 9.0.5" and the "Installer" button is visible. The "Description" section states: "Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio. Enables these commonly used commands: Add-Migration, Bundle-Migration, Drop-Database, Get-DbContext, Get-Migration, Optimize-DbContext, ...".

On the right side of the interface, the Solution Explorer shows the project "EasyApp" with files like AppDbContext.cs, Faculte.cs, Etudiant.cs, MainWindow.xaml, and XamlReader.cs. The Properties window is also visible.

Une fois sélectionné, cliquez sur installer

Ass. Eric Wangi  
243812961556



Ass. Eric Wangi  
243812961556

## Ajoutons l'entité Framework pour SQL Serveur car notre SGBD serait MS SQL Serveur:

Fichier Edition Affichage Git Projet Générer Déboguer Test Analyser Outils Extensions Fenêtre Aide Rechercher EasyApp Connexion GitHub Copilot

EasyApp\* NuGet : EasyApp AppDbContext.cs\* Faculte.cs Etudiant.cs MainWindow.xaml XamlReader.cs

Parcourir Installé Mises à jour Entityframework sql server Inclure la version préliminaire

Gestionnaire de package NuGet : EasyApp Source de package : nuget.org

Microsoft.EntityFrameworkCore.SqlServer par aspnet, dotnetframework, EntityFramework 9.0.5 Microsoft SQL Server database provider for Entity Framework Core.

Microsoft.EntityFrameworkCore par aspnet, dotnetframework, EntityFramework 9.0.5 Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with...

Z.EntityFramework.Extensions.EFCore par zzzprojects, 49,6M téléchargé 9.103.8.1 Microsoft.EntityFrameworkCore Extension Methods

Chaque package vous est concédé sous licence par son propriétaire. NuGet n'est pas responsable des packages tiers et n'octroie aucune licence les concernant.

Ne plus afficher ceci

Liste d'erreurs 0 Erreurs 4 Avertissements 0 de 7 Messages Build + IntelliSense Rechercher dans la liste des erreurs

Code Description Le propriété « nom » non-nullab... Fichier

Le propriété « nom » non-nullab... Accédez aux modifications

Explorateur de solutions Solution 'EasyApp' (1 sur 1 de projets)

EasyApp Dépendances ApplicationDBContexte AppDbContext.cs models App.xaml AssemblyInfo.cs MainWindow.xaml

Outils de diagnostic Propriétés



Ass. Eric Wangi  
243812961556

Revenons à notre classe **AppDbContext** pour la configuration de la notre base de données:

```
✓ using System;
  using System.Collections.Generic;
  using System.Linq;
  using System.Text;
  using System.Threading.Tasks;
  using EasyApp.models;
  using Microsoft.EntityFrameworkCore;

✓ namespace EasyApp.ApplicationDBContexte
{
    0 références
    ✓ public class AppDbContext : DbContext
    {
        0 références
        ✓ public DbSet<Etudiant> Etudiant { get; set; }
        0 références
        ✓ public DbSet<Faculte> Faculte { get; set; }

        0 références
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
            if (!optionsBuilder.IsConfigured)
            {
                optionsBuilder.UseSqlServer(@"Data source=DESKTOP-22C71FD;
Initial Catalog=GestionEtudiant;TrustServerCertificate=True;Integrated Security=True");
            }
        }
    }
}
```

Les **DBSet** sont des classes qui doivent se traduire en table



Ass. Eric Wangi  
243812961556

```
0 références
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Etudiant>()
        .HasOne(et => et.faculte)
        .WithMany(fa => fa.etudiants)
        .HasForeignKey(et => et.faculteId);
}
```

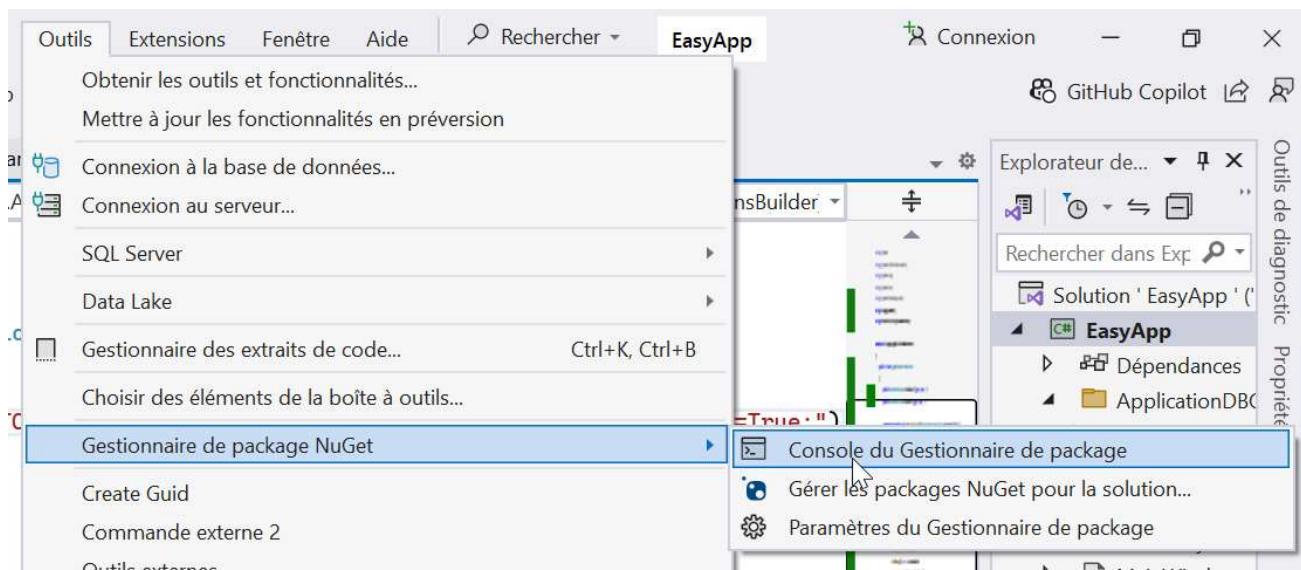
NB:

Cette méthode **OnModelCreating** est utilisée dans Entity Framework Core pour configurer les relations entre les entités du modèle de données.



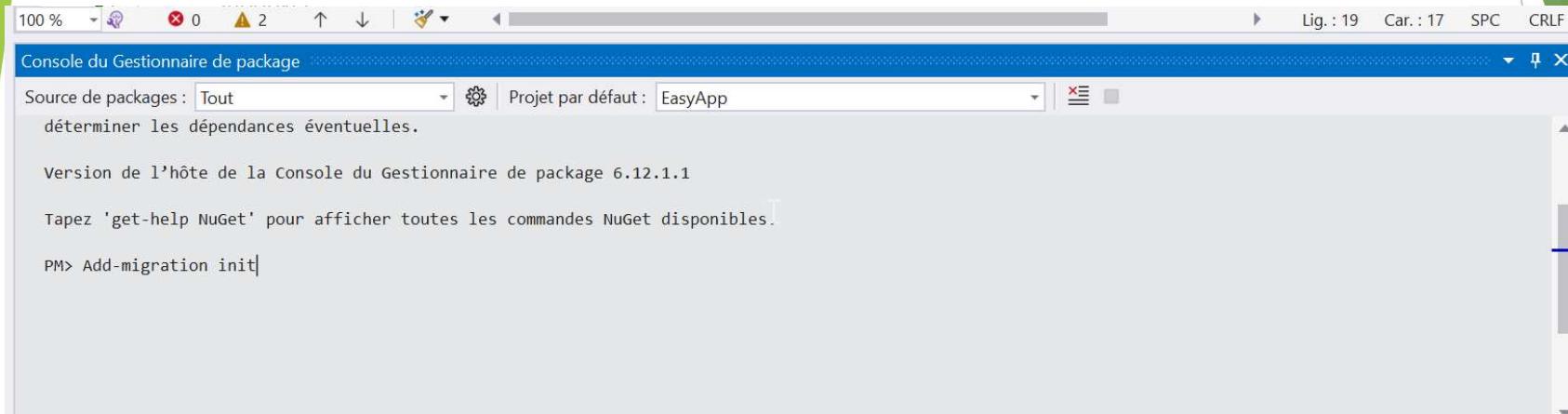
Ass. Eric Wangi  
243812961556

## 4.6.2. La Migration



Ass. Eric Wangi  
243812961556

Tapons la commande **Add-migration suivi\_nom\_migration.**



Console du Gestionnaire de package

Source de packages : Tout | Projet par défaut : EasyApp

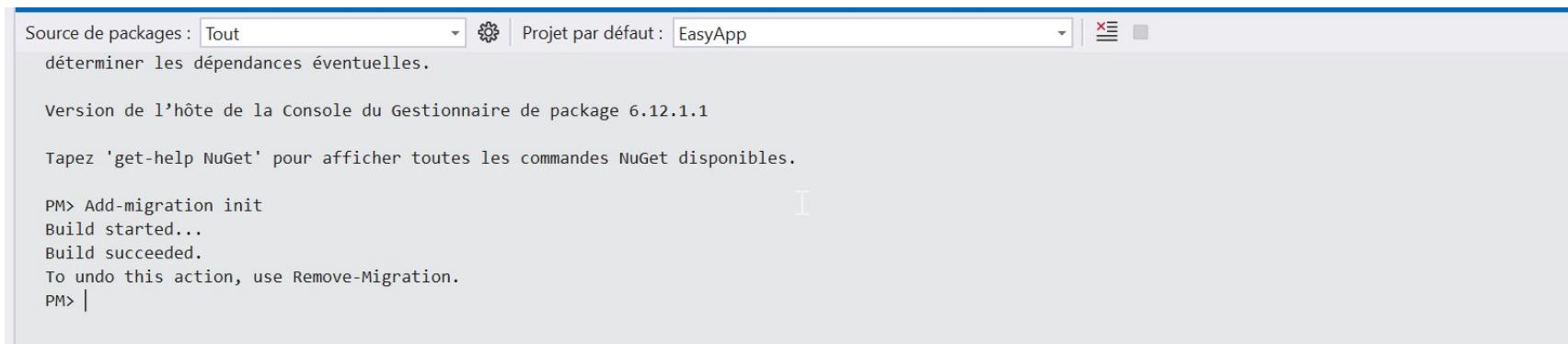
déterminer les dépendances éventuelles.

Version de l'hôte de la Console du Gestionnaire de package 6.12.1.1

Tapez 'get-help NuGet' pour afficher toutes les commandes NuGet disponibles.

PM> Add-migration init|

Appuyez sur la touche Entrer.



Source de packages : Tout | Projet par défaut : EasyApp

déterminer les dépendances éventuelles.

Version de l'hôte de la Console du Gestionnaire de package 6.12.1.1

Tapez 'get-help NuGet' pour afficher toutes les commandes NuGet disponibles.

PM> Add-migration init

Build started...

Build succeeded.

To undo this action, use Remove-Migration.

PM> |



Ass. Eric Wangi  
243812961556

Le fichier de migration vient d'être créé:

```
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace EasyApp.Migrations
{
    /// <inheritdoc />
    1 référence
    public partial class init : Migration
    {
        /// <inheritdoc />
        0 références
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Faculte",
                columns: table => new
                {
                    id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    faculteId = table.Column<int>(type: "int", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Faculte", x => x.id);
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Faculte");
        }
}
```



Ass. Eric Wangi  
243812961556

```

migrationBuilder.CreateTable(
    name: "Etudiant",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        nom = table.Column<string>(type: "nvarchar(max)", nullable: false),
        prenom = table.Column<string>(type: "nvarchar(max)", nullable: false),
        faculteId = table.Column<int>(type: "int", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Etudiant", x => x.Id);
        table.ForeignKey(
            name: "FK_Etudiant_Faculte_faculteId",
            column: x => x.faculteId,
            principalTable: "Faculte",
            principalColumn: "id",
            onDelete: ReferentialAction.Cascade);
    });
}

migrationBuilder.CreateIndex(
    name: "IX_Etudiant_faculteId",
    table: "Etudiant",
    column: "faculteId");
}

/// <inheritdoc />
0 références
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Etudiant");

    migrationBuilder.DropTable(
        name: "Faculte");
}
}

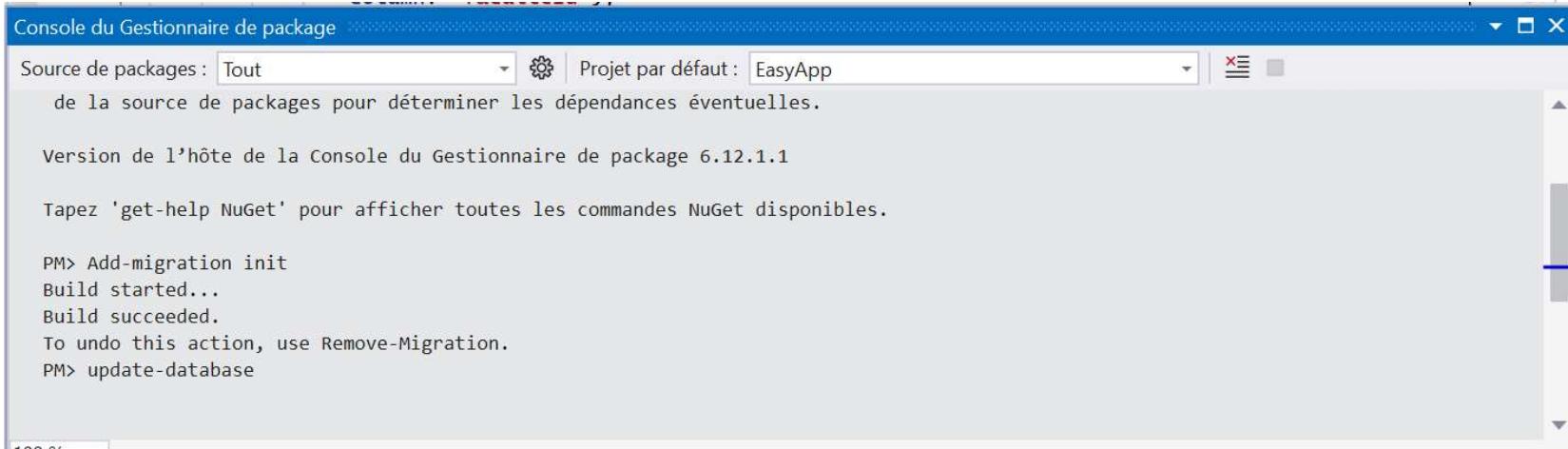
```



Ass. Eric Wangi  
243812961556

#### 4.6.3. Création de la base de données

Pour que la migration soit réellement appliquée au niveau de la base de données, il suffit de faire **update-database**.



The screenshot shows the NuGet Package Manager Console window. The title bar says "Console du Gestionnaire de package". The interface includes dropdown menus for "Source de packages" (set to "Tout") and "Projet par défaut" (set to "EasyApp"). Below the menu bar, there is a message: "de la source de packages pour déterminer les dépendances éventuelles." The console output area displays the following text:

```
Version de l'hôte de la Console du Gestionnaire de package 6.12.1.1
Tapez 'get-help NuGet' pour afficher toutes les commandes NuGet disponibles.

PM> Add-migration init
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> update-database
```



Ass. Eric Wangi  
243812961556

Console du Gestionnaire de package

Source de packages : Tout | Projets | Projet par défaut : EasyApp | X

```
PM> update-database
PM> Build started...
Build succeeded.
Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
Applying migration '20250526153021_init'.
Done.
PM> |
```

100 %

Sortie Console du Gestionnaire de package

Explorateur d'objets

Connecter ▾

DESKTOP-22C71FD (SQL Server 13.0)

- Bases de données
  - Bases de données système
  - Instantanés de base de données
  - GestionEtudiant
    - Schémas de base de données
    - Tables
      - Tables système
      - FileTables
      - Tables externes
    - dbo.\_EFMigrationsHistory
    - dbo.Etudiant
    - dbo.Faculte
  - Vues
  - Ressources externes
  - Synonymes
  - Programmabilité
  - Service Broker
  - Stockage
  - Sécurité

Ass. Eric Wangi  
243812961556

## 4.7.ViewModel

Le ViewModel est une classe utilisée dans le modèle de conception MVVM.

Il sert d'intermédiaire entre le **Model** (données, logique métier) et la **View** (interface utilisateur).

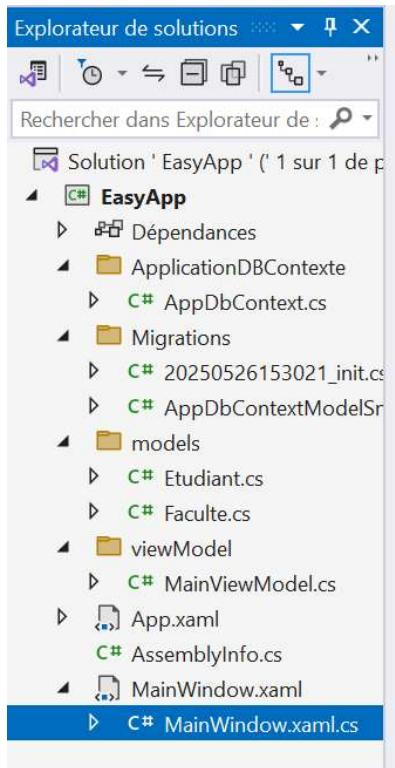
Son objectif principal est de :

- 1. Encapsuler la logique de présentation** : Il transforme les données du modèle pour les rendre exploitables par la vue.
- 2. Gérer les interactions utilisateur** : Il réagit aux actions de l'utilisateur et met à jour les données en conséquence.
- 3. Favoriser le découplage** : Il sépare strictement la logique métier de l'affichage, ce qui facilite les tests unitaires et la maintenance.



Ass. Eric Wangi  
243812961556

Créons un nouveau dossier nommé viewModel.



Ajoutons maintenant une classe nommée **MainViewModel.cs**.



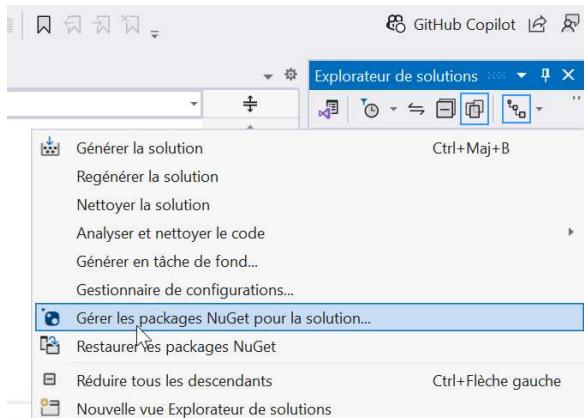
Ass. Eric Wangi  
243812961556

```
✓ using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

✓ namespace EasyApp.viewModel
{
    ✓ class MainViewModel
    {
    }
}
```

Nous allons la rendre public. Nous devons aussi étendre notre **ViewModel** de la classe **ObservableObject** pour qu'elle devient observable.

Veuillez d'abord installer ce package dans le projet.



Ass. Eric Wangi  
243812961556

NuGet - Solution MainViewModel.cs

Parcourir      Installé      Mises à jour      Consolider

CommunityToolkit    Inclure la version préliminaire

Gérer les packages de la solution

Source de package : nuget.org

**CommunityToolkit.HighPerformance** par dotnetfoundation, Microsoft.Toolkit 8.4.0

This package includes high performance .NET helpers such as:

- Memory2D<T> and Span2D<T>: two types providing fast and allocation-free abst...

**CommunityToolkit.Mvvm** par dotnetfoundation, Microsoft.Toolkit, 11,5M téléchargé 8.4.0

This package includes a .NET MVVM library with helpers such as:

- ObservableObject: a base class for objects implementing the INotifyPropertyChan...

**CommunityToolkit.Diagnostics** par dotnetfoundation, Microsoft.Toolkit, 7,55M 8.4.0

This package includes .NET helpers such as:

- Guard: Helper methods to verify conditions when running code.

**CommunityToolkit.Maui.Core** par dotnetfoundation, Microsoft.Toolkit, 4,55M 11.2.0

Core library for community toolkits using .NET MAUI

**CommunityToolkit.Maui** par dotnetfoundation, Microsoft.Toolkit, 4,62M téléchargé 11.2.0

Chaque package vous est concédé sous licence par son propriétaire. NuGet n'est pas responsable des

CommunityToolkit.Mvvm nuget.org

Versions : 0

<input checked="" type="checkbox"/> Projet	<input type="checkbox"/> Version	<input type="checkbox"/> Installée	<input type="checkbox"/> Niveau du package
<input checked="" type="checkbox"/> EasyApp			

Installé : non installé

Version : Dernière version stable 8.4.0

Le mappage de la source du paquet est désactivé.



Cliquez sur installer

Ass. Eric Wangi  
243812961556

NuGet - Solution MainViewModel.cs

Parcourir Installé Mises à jour Consolider

CommunityToolkit

Inclure la version préliminaire

Gérer les packages de la solution

Source de package : nuget.org

CommunityToolkit.HighPerformance par dotnetfoundation, Microsoft.Toolkit 8.4.0

This package includes high performance .NET helpers such as:  
- Memory2D<T> and Span2D<T>: two types providing fast and allocation-free abst...

CommunityToolkit.Mvvm par dotnetfoundation, Microsoft.Toolkit, 11,5M téléchargé 8.4.0

This package includes a .NET MVVM library with helpers such as:

Sortie

Afficher la sortie à partir de : Gestionnaire de package

Restauration des packages pour C:\Users\ERIWANG\Documents\projetcsharp\EasyApp\EasyApp.csproj...

GET <https://api.nuget.org/v3-flatcontainer/communitytoolkit.mvvm/index.json> 603 ms

OK <https://api.nuget.org/v3-flatcontainer/communitytoolkit.mvvm/index.json>

GET <https://api.nuget.org/v3-flatcontainer/communitytoolkit.mvvm/8.4.0/communitytoolkit.mvvm.8.4.0.nupkg> 178 ms

OK <https://api.nuget.org/v3-flatcontainer/communitytoolkit.mvvm/8.4.0/communitytoolkit.mvvm.8.4.0.nupkg>

Explorateur de solutions

Solution 'EasyApp' (1 sur 1 de p)

- EasyApp
  - Dépendances
  - ApplicationDbContext
    - AppDbContext.cs
  - Migrations
    - 20250526153021\_init.cs
    - AppDbContextModelSnapshot.cs
  - models

Activer Windows

Accédez aux paramètres pour activer Windows.

Prêt

13:04

29°C Très ensoleillé



Ass. Eric Wangi  
243812961556

Nous pouvons maintenant étendre notre classe de **ObservableObject** se trouvant dans **CommunityToolkit.Mvvm.ComponentModel** qu'il faut importer.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CommunityToolkit.Mvvm.ComponentModel;

namespace EasyApp.viewModel
{
    class MainViewModel: ObservableObject
    {
    }
}
```

Pour communiquer notre **viewModel** à la page réalisée précédemment(**MainWindow.xaml.cs**), nous allons faire:



Ass. Eric Wangi  
243812961556

```
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 EasyApp.viewModel;

namespace EasyApp
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 références
    public partial class MainWindow : Window
    {
        0 références
        public MainWindow()
        {
            InitializeComponent();
            DataContext = new MainViewModel();
        }
    }
}
```

NB. N'est pas oublier d'importer le **viewModel** que nous venons de créer directement dans le projet.



Ass. Eric Wangi  
243812961556

Revenons au MainViewModel pour définir :

1. définir les propriétés
2. Une liste des facultés
3. Les propriétés public des nos composants pour pouvoir récupérer le contenu du champ nom, prénom si cela change dans la zone de texte.
4. Les constructeurs. Nous allons aussi profiter pour créer des instances de chaque liste que nous avons déclaré précédemment. En rappel, pour que les contenus de nos listes se mettent automatiquement à jour lorsque l'un d'entre eux est mis à jour, il serait mieux que nous l'instancions comme des **ObservableCollections**. De préférence, changeons aussi au niveau 2



Ass. Eric Wangi  
243812961556

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CommunityToolkit.Mvvm.ComponentModel;
using EasyApp.models;

namespace EasyApp.viewModel
{
    2 références
    class MainViewModel : ObservableObject
    {
        private string nom;
        private string prenom;
        private Faculte faculteSelected;

        1 référence
        public ObservableCollection<Faculte> facultes { get; set; }
        1 référence
        public ObservableCollection<Etudiant> etudiants { get; set; }

        1 référence

        public string Nom {
            get => nom;
            set {
                SetProperty(ref nom, value);
                OnPropertyChanged(nameof(Nom));
            }
        }

        1 référence
        public string Prenom
        {
            get => prenom;
            set { SetProperty(ref prenom, value);
                OnPropertyChanged(nameof(Prenom));
            }
        }
    }
}
```



Ass. Eric Wangi  
243812961556

```

1 référence
public Faculte FaculteSelected
{
    get => faculteSelected;
    set
    {
        SetProperty(ref faculteSelected, value);
        OnPropertyChanged(nameof(FaculteSelected));
    }
}

1 référence
public MainViewModel()
{
    facultes = new ObservableCollection<Faculte>();
    etudiants = new ObservableCollection<Etudiant>();
}
}

```

### Remarque:

Nous avons déclaré des propriétés publique ayant à la fois un accesseur et un mutateur.

Au niveau du mutateur la méthode **SetProperty** est appelée lorsqu'une valeur doit être mise à jour.

La méthode **OnPropertyChanged**, déclenche un événement de notification pour signaler que la valeur de la propriété a changé. Cela est couramment utilisé dans le pattern MVVM pour mettre à jour l'interface utilisateur de manière réactive.



Ass. Eric Wangi  
243812961556

Créons maintenant une méthode **LoadData** qui permet de charger les données et les affichées.

Avant tout, nous devons faire appel au **AppDbContext** car celui qui connaît nos models puis l'initialiser dans le constructeur.

```
class MainViewModel:ObservableObject
{
    private readonly AppDbContext context;
```



Ass. Eric Wangi  
243812961556

La méthode **LoadData** se présente comme suit:

```
1 référence
public void LoadData()
{
    //On récupère tous les etudiants dans la BD sous forme d'une liste
    //et les affectés dans une nouvelle variable etudiantList
    var etudiantList = context.Etudiant.ToList();
    //On ajouter les etudiantList dans notre liste etudiants |
    foreach(var et in etudiantList)
    {
        etudiants.Add(et);
    }
}
```

NB. N'est pas oublié d'appeler la méthode **LoadData** dans le constructeur.

```
1 référence
public MainViewModel()
{
    context = new AppDbContext();
    facultes = new ObservableCollection<Faculte>();
    etudiants = new ObservableCollection<Etudiant>();
    LoadData();
}
```



Ass. Eric Wangi  
243812961556

Allons dans la vue **MainWindow.xaml.cs** pour aller au lieu d'emplacement de la Liste des étudiants :

```
<!-- Affichage d'une grille pour la liste des étudiants -->
<ListView Grid.Row="1">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Nom" Width="250"/>
            <GridViewColumn Header="Prenom" Width="100"/>
            <GridViewColumn Header="Faculté" Width="100"/>
        </GridView>
    </ListView.View>
</ListView>
```

Faisons appel à la propriété **ItemSource** pour faire un Binding avec la liste **etudiants(attribut)** créée dans *MainViewModel*. Pour finir nous allons dire au niveau de chaque entête le nom de champs de la BD (ou Model) à afficher en respectant la case se trouvant dans la BD ou encore dans le model réalisé:

```
<!-- Affichage d'une grille pour la liste des étudiants -->
<ListView Grid.Row="1"
          ItemsSource="{Binding etudiants}">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Nom" Width="250" DisplayMemberBinding="{Binding nom}"/>
            <GridViewColumn Header="Prenom" Width="100" DisplayMemberBinding="{Binding prenom}"/>
            <GridViewColumn Header="Faculté" Width="100" DisplayMemberBinding="{Binding faculteId}"/>
        </GridView>
    </ListView.View>
</ListView>
```

Attribut du type  
**ObservableCollection**  
déclaré dans  
*MainViewModel*



Ass. Eric Wangi  
243812961556

Testons:

Nous allons alimenter provisoirement manuellement nos deux tables dans la BD:

	id	faculteid
▶	1	1
	2	2

	Id	nom	prenom	faculteid
▶	2	WANGI NGOY	ERIC	1
*	NULL	NULL	NULL	NULL



Ass. Eric Wangi  
243812961556

mainwindow.exe

MainWindow

Nom

Prenom

Faculté

Enregistrer   Modifier   Supprimer

Nom	Prenom	Faculté
WANGI NGOY	ERIC	1



Ass. Eric Wangi  
243812961556

Nous devons modifier notre requête de sorte que pour chaque Etudiant chargé, on vient avec sa faculté:

```
    public void LoadData()
    {
        //On récupère tous les etudiants dans la BD sous forme d'une liste
        //et les affectés dans une nouvelle variable etudiantList
        var etudiantList = context.Etudiant
            .Include(p=>p.faculte)
            .ToList();
        //On ajouter les étudiantList dans notre liste etudiants
        foreach(var et in etudiantList)
        {
            etudiants.Add(et);
        }
    }
```

NB. Au départ la table Faculté n'avait pas un nom. Veuillez modifier le model faculté puis faire la migration

Au niveau de MainWindows.xaml.cs:

```
<ListView Grid.Row="1"
          ItemsSource="{Binding etudiants}">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Nom" Width="250" DisplayMemberBinding="{Binding nom}" />
            <GridViewColumn Header="Prenom" Width="100" DisplayMemberBinding="{Binding prenom}" />
            <GridViewColumn Header="Faculté" Width="100" DisplayMemberBinding="{Binding faculte.nomFaculte}" />
        </GridView>
    </ListView.View>

</ListView>
```



Ass. Eric Wangi  
243812961556

MainWindow

Nom

Prenom

Faculté

Enregistrer   Modifier   Supprimer

Nom	Prenom	Faculté
WANGI NGOY	ERIC	FASI



Ass. Eric Wangi  
243812961556

## 4.8. Ajout d'une donnée dans la BD

Toujours dans **MainViewModel.cs**, nous allons créer une méthode Ajouter:

```
[RelayCommand]
2 références
private void Ajouter()
{
    try
    {
        //On récupère les valeur à partir de set
        var _nom = Nom;
        var _prenom = Prenom;
        var _fac = FaculteSelected.id;
        //affectation des valeur dans l'objet etudiant
        var etudiant = new Etudiant
        {
            nom = _nom,
            prenom = _prenom,
            faculteId = _fac
        };
        //Mise à jour
        context.Etudiant.Add(etudiant);
        context.SaveChanges();
        //Message de confirmation
        MessageBox.Show("Opération réussie", "Information",
            MessageBoxButton.OK, MessageBoxImage.Information);
        //Chargement des données et réinitialisation du formulaire
        LoadData();
        Nom = "";
        Prenom = "";
        FaculteSelected = null;
    }
    catch (Exception ex) {
        MessageBox.Show("Problème rencontré:" + ex.Message, "Erreur",
            MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```



Ass. Eric Wangi  
243812961556

Rentrons dans **MainWindows.xaml.cs** pour appeler la méthode Ajouter suivi du mot Command lorsque le bouton Enregistrer sera cliqué:

```
<Button Content="Enregistrer" Background="Blue" Command="{Binding AjouterCommand}"/>
```



Ass. Eric Wangi  
243812961556

Au niveau de champ de saisie ajoutons:

```
<TextBlock Text="Nom"/>
<TextBox Text="{Binding Nom}"/>
<TextBlock Text="Prenom"/>
<TextBox Text="{Binding Prenom}"/>
```

Voir les accesseurs  
crées dans  
MainViewModel

- **TextBlock** joue le rôle d'un label destiné à afficher un texte statique chez l'utilisateur.
- **TextBox** est un composant permettant à un utilisateur de saisir un texte ou affichera une valeur.
- **{Binding Nom}** crée une liaison (binding) avec la propriété Nom définie dans le **ViewModel**.
- Si la propriété Nom change, le TextBox mettra automatiquement à jour son affichage.



Ass. Eric Wangi  
243812961556

Au niveau de champ de saisie ajoutons:

```
<TextBlock Text="Nom"/>
<TextBox Text="{Binding Nom}"/>
<TextBlock Text="Prenom"/>
<TextBox Text="{Binding Prenom}"/>
```

Voir les accesseurs  
crées dans  
MainViewModel

signifie que le **TextBlock** affichera la valeur de la propriété Nom du **DataContext** actuel.

- **{Binding Nom}** crée une liaison (binding) avec la propriété Nom définie dans le **ViewModel**.
- Si la propriété Nom change, le TextBox mettra automatiquement à jour son affichage.

Comme nous n'avons pas les données dans la table faculté, il serait mieux d'insérer quelques données. Allons dans MainViewModel pour ajouter la ligne :

```
1 référence
public void ajouterFaculte()
{
    context.Faculte.AddRange(
        new Faculte { nomFaculte = "FASI" },
        new Faculte { nomFaculte = "DROIT" }
    );
    context.SaveChanges();
}
```



NB. N'est pas oublier de l'appeler dans le constructeur de **MainViewModel.cs** de telle sorte que les facultés soient créées lors du changement du formulaire

Ass. Eric Wangi  
243812961556

Par contre, nous pouvons maintenant modifier notre liste de choix de sorte que les données proviennent de la base de données.

Dans MainWindows.xaml.cs modifions comme suit:

```
public void LoadData()
{
    etudiants.Clear();
    //On récupère tous les etudiants dans la BD sous forme d'une liste
    //et les affectés dans une nouvelle variable etudiantList
    var etudiantList = context.Etudiant
        .Include(p => p.faculte)
        .ToList();
    //On ajouter les etudiantList dans notre liste etudiants
    foreach(var et in etudiantList)
    {
        etudiants.Add(et);
    }
    //Ajout des facultés dans la liste
    var faculteList = context.Faculte
        .ToList();
    foreach (var fac in faculteList)
    {
        facultes.Add(fac);
    }
}
```



Ass. Eric Wangi  
243812961556

Au niveau du constructeur :

```
1 référence
public MainViewModel()
{
    context = new AppDbContext();
    facultes = new ObservableCollection<Faculte>();
    etudiants = new ObservableCollection<Etudiant>();

    ajouterFaculter();
    LoadData();
}
```

Allons dans la vue MainWindow.xaml.cs pour modifier l'affichage du comboBox:

```
<ComboBox ItemsSource="{Binding facultes}"
          DisplayMemberPath="nomFaculte"
          SelectedValuePath="id"
          SelectedItem="{Binding FaculteSelected}">

```

Attribut du type  
**ObservableCollection**  
déclaré dans  
MainViewModel.cs

Propriété publique du  
type Faculté ayant le  
**get et set** déclaré dans  
MainViewModel.cs



Ass. Eric Wangi  
243812961556

## 4.9. Modification d'une donnée dans la BD

Nous allons d'abord vérifier si un étudiant est sélectionné. Pour cela, il faut un champs supplémentaire nommé etudiantSelected du type Etudiant.

```
private Etudiant etudiantSelected;
```

Créons une propriété pour etudiantSelected:

```
1 référence
public Etudiant EtudiantSelected
{
    get => etudiantSelected;
    set
    {
        if (etudiantSelected != value)
        {
            SetProperty(ref etudiantSelected, value);
            OnPropertyChanged(nameof(EtudiantSelected));

            if (etudiantSelected != null)
            {
                Nom = etudiantSelected.nom;
                Prenom = etudiantSelected.prenom;
                FaculteSelected = facultes.FirstOrDefault(fac => fac.id == etudiantSelected.Id);
            }
        }
    }
}
```

A partir de la liste de faculté, nous allons comparé les identifiants avec l'identifiant se trouvant chez l'étudiant sélectionné. Dans le cas d'égalité, le détail de la faculté est affiché dans la liste de choix.

Nous devons dire maintenant où est ce que l'étudiant sera sélectionné. Comme nous le savons cela serait fait dans la listView. Pour cela, nous devons modifier la listView comme suit:

```
<!-- Affichage d'une grille pour la liste des étudiants -->
<ListView Grid.Row="1"
          ItemsSource="{Binding etudiants}"
          SelectedItem="{Binding EtudiantSelected}">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Nom" Width="250" DisplayMemberBinding="{Binding nom}"/>
            <GridViewColumn Header="Prenom" Width="100" DisplayMemberBinding="{Binding prenom}"/>
            <GridViewColumn Header="Faculté" Width="100" DisplayMemberBinding="{Binding faculte.nomFaculte}"/>
        </GridView>
    </ListView.View>
</ListView>
```

MainWindow

Nom  
WANGI NGOY

Prenom  
ERIC

Faculté  
DROIT

**Enregistrer** **Modifier** **Supprimer**

Nom	Prenom	Faculté
WANGI NGOY	ERIC	FASI
MAK	MAT	MEDECINE
MAKAMPILA	JOELLE	FASI
DEO	MAS	FASI
zeze	tijo	FASI

Ass. Eric Wangi  
243812961556

Maintenant nous pouvons créer une méthode pour la modification:

```
[RelayCommand]
2 références
private void Modifier()
{
    try
    {

        if (EtudiantSelected != null)
        {
            //On récupère les valeur à partir de set
            EtudiantSelected.nom = Nom;
            EtudiantSelected.prenom = Prenom;
            EtudiantSelected.faculteId = FaculteSelected.id;
            context.SaveChanges();

            //Message de confirmation
            MessageBox.Show("Opération réussie", "Information",
                MessageBoxButtons.OK, MessageBoxIcon.Information);

            //Chargement des données et réinitialisation du formulaire
            LoadData();
            Nom = "";
            Prenom = "";
            FaculteSelected = null;
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show("Problème rencontré:" + ex.Message, "Erreur",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

```
<Button Content="Modifier" Background="Green" Command="{Binding ModifierCommand}"/>
```

Ass. Eric Wangi  
243812961556

## 4.10. Suppression d'une donnée dans la BD

```
[RelayCommand]
2 références
private void Supprimer()
{
    try
    {
        if (EtudiantSelected != null)
        {
            // Vérification si l'étudiant existe bien dans le contexte avant suppression
            var etudiantASupprimer = context.Etudiant.FirstOrDefault(e => e.Id == EtudiantSelected.Id);

            if (etudiantASupprimer != null)
            {
                context.Etudiant.Remove(etudiantASupprimer);
                context.SaveChanges();

                // Message de confirmation
                MessageBox.Show("Opération réussie", "Information",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);

                // Chargement des données et réinitialisation du formulaire
                LoadData();
                Nom = string.Empty;
                Prenom = string.Empty;
            }
        }
    }
}
```

```
        else
    {
        MessageBox.Show("L'étudiant sélectionné n'existe pas dans la base de données.", "Erreur",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
    else
    {
        MessageBox.Show("Veuillez sélectionner un étudiant avant de supprimer.", "Avertissement",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
catch (Exception ex)
{
    MessageBox.Show("Problème rencontré : " + ex.Message, "Erreur",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

```
<Button Content="Supprimer" Background="Red" Command="{Binding SupprimerCommand}"/>
```

Ass. Eric Wangi  
243812961556