

Cours de C#

I. Rappels POO

- POO vs Procedural
- Classe vs Objet
- Relation
 - Héritage
 - Polymorphisme
 - Bidirectionnelle ou Unidirectionnelle
 - OneToMany
 - ManyToMany
 - ManyToOne
 - OneToOne
- Interface
- Liste
- Inversion de Control
- Couplage Fort Vs Couplage Faible
- Injection de Dépendance

I. Projet Fil Rouge

L'Institut Supérieur de Management ISM fait appel à vous pour la réalisation d'une application Desktop JavaFX de gestion des inscriptions.

Chaque début d'année le Responsable Pédagogique peut créer,lister des classes (libellé).Il a la possibilité aussi d'ajouter des professeurs et leurs affectés des classes,lister les professeurs(Nci,nom complet,grade) et filtrer les classes d'un professeur.

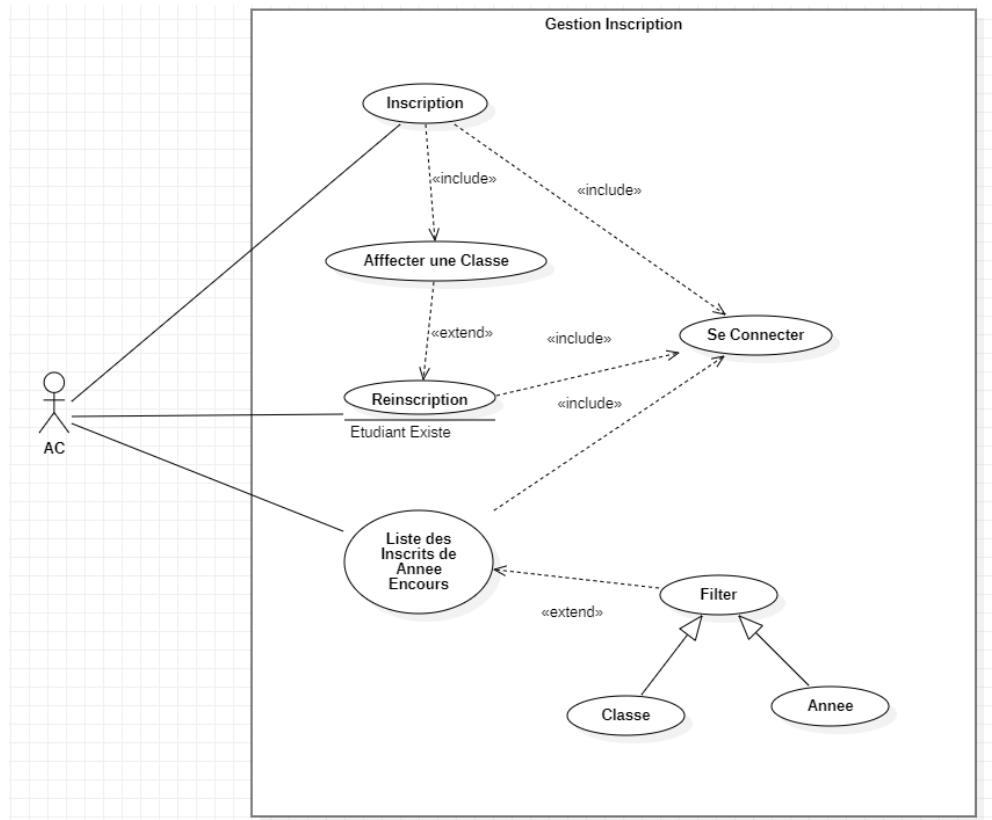
Les Attachés de classe font les inscriptions et les réinscriptions des étudiants durant la période d' inscription.Un étudiant(matricule,nom complet,tuteur) peut s'inscrire plusieurs fois mais une seule fois dans une année.

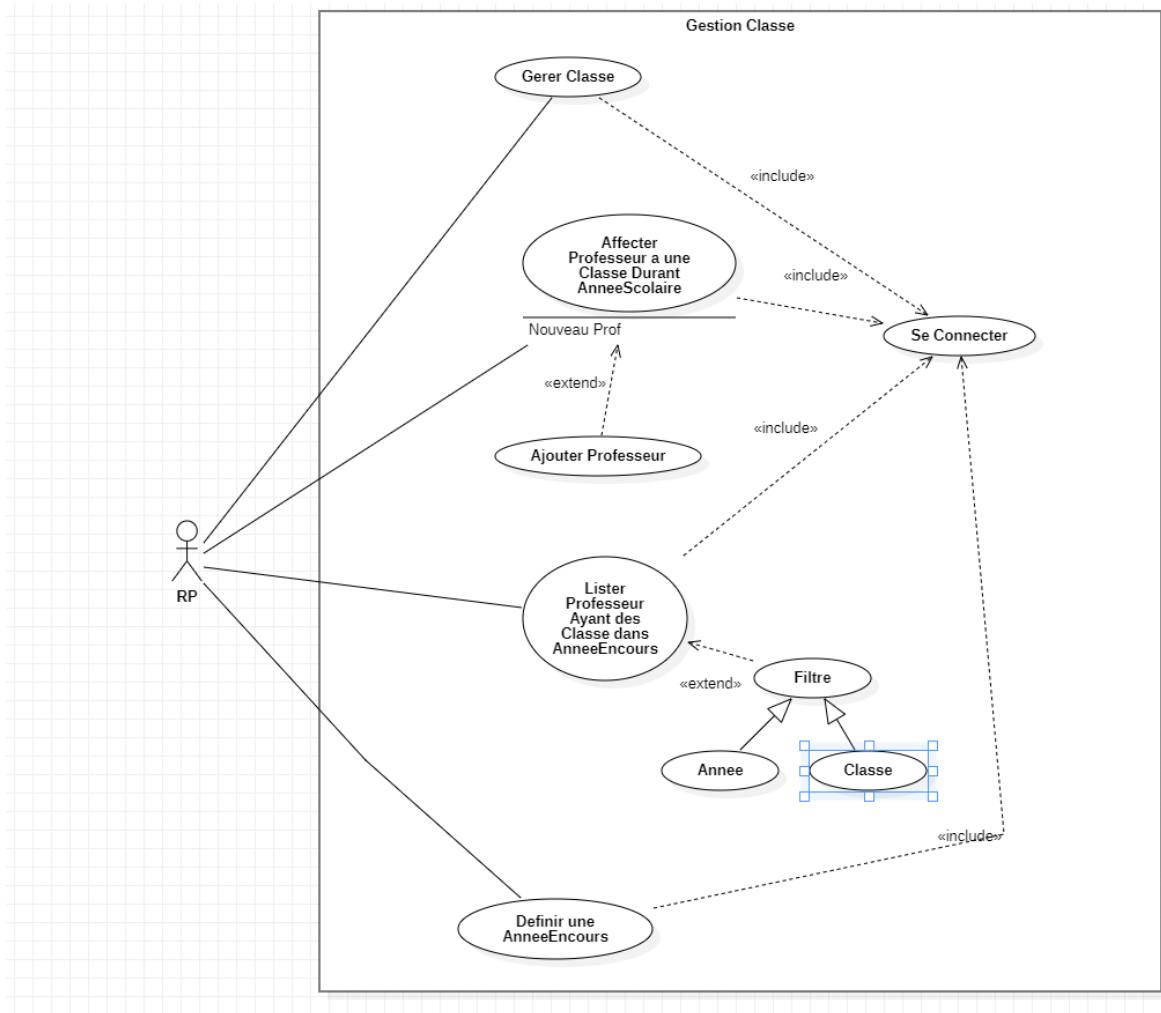
Un Attaché a la possibilité de lister les étudiants inscrits dans une année,de filtrer cette liste par classe.

NB: Toutes les Fonctionnalités sont accessibles après Connexion

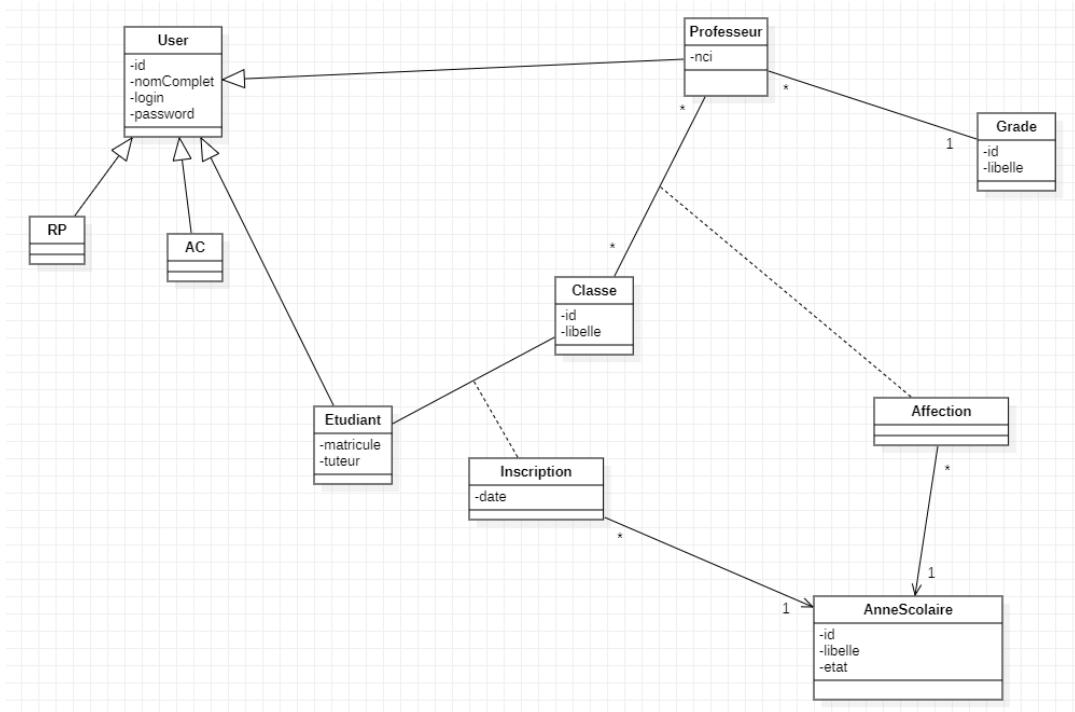
II. Modélisation

- Diagramme Use Case





• Diagramme de Classe



- MLD

III. Architecture MVP

A. Présentation

B. Principe

MVP est le sigle pour MODÈLE - VUE - PRÉSENTATION.

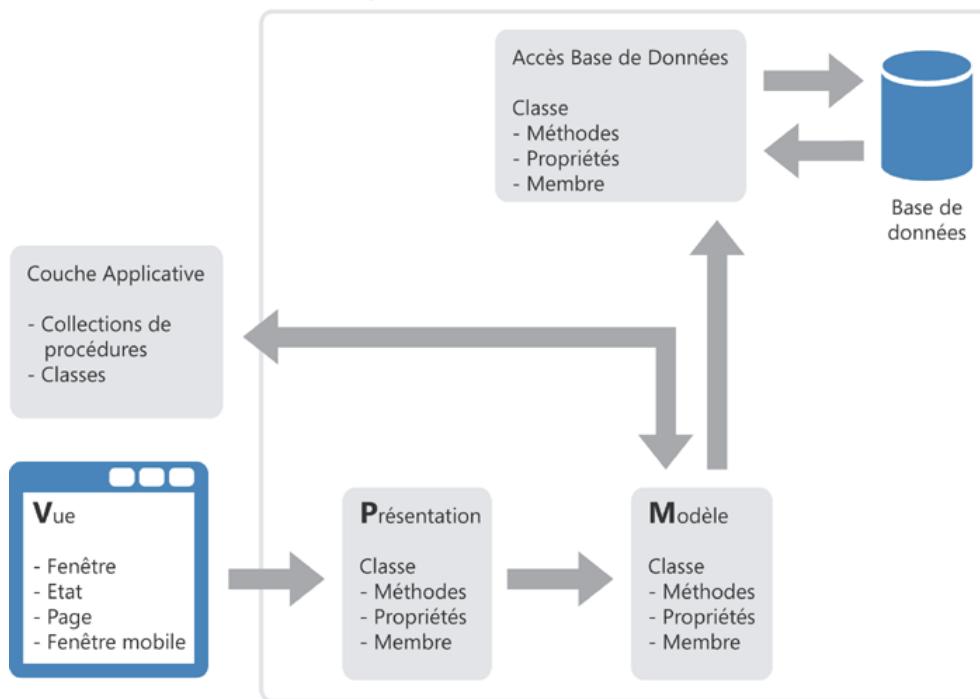
C'est un "Design Pattern" qui propose de découper et de structurer l'architecture des interfaces utilisateur en couches. Il permet de séparer le code de gestion de l'interface (UI), du code qui manipule les données métier.

Cette architecture en couches de l'application et de l'UI permet de rendre les projets plus faciles à maintenir et à faire évoluer.

Un modèle de développement MVP se décompose en couches comme suit :

1. **Une couche PRÉSENTATION.**
2. **Une couche MODÈLE.**
3. **Une couche Accès BDD (Base De Données) ou Repositories.**
4. **Une couche Applicative ou Service.**

Principe MVP



5. La couche VUE.

Une VUE représente la partie UI de l'application. Elle correspond à l'interface utilisateur (UI). Une VUE peut être sous la forme d'une fenêtre.

Certaines opérations peuvent nécessiter une interaction avec l'utilisateur (affichage d'erreur, confirmation, ...), ces interactions doivent être assurées par la VUE. La VUE **dépend** de la couche PRÉSENTATION.

La VUE peut utiliser le binding pour récupérer les données à afficher depuis la couche PRÉSENTATION, ou lui envoyer les informations saisies ou modifiées par l'utilisateur.

La VUE dispose d'événements particuliers pour effectuer les mises à jour nécessaires .

6. La couche PRÉSENTATION

La couche Présentation est une classe qui effectue la liaison entre la VUE et les Services. Elle regroupe les traitements concernant les actions de l'utilisateur.

La couche PRESENTATION n'a pas accès à la VUE, c'est-à-dire que la couche PRÉSENTATION ne doit pas accéder directement aux champs de la VUE .

Par contre, la couche PRÉSENTATION peut demander à la VUE de se mettre à jour .

La couche PRESENTATION contient et décide des "données" à afficher dans la VUE. Par exemple, changer l'état d'un champ, changer la couleur d'une ligne d'un champ Table,...

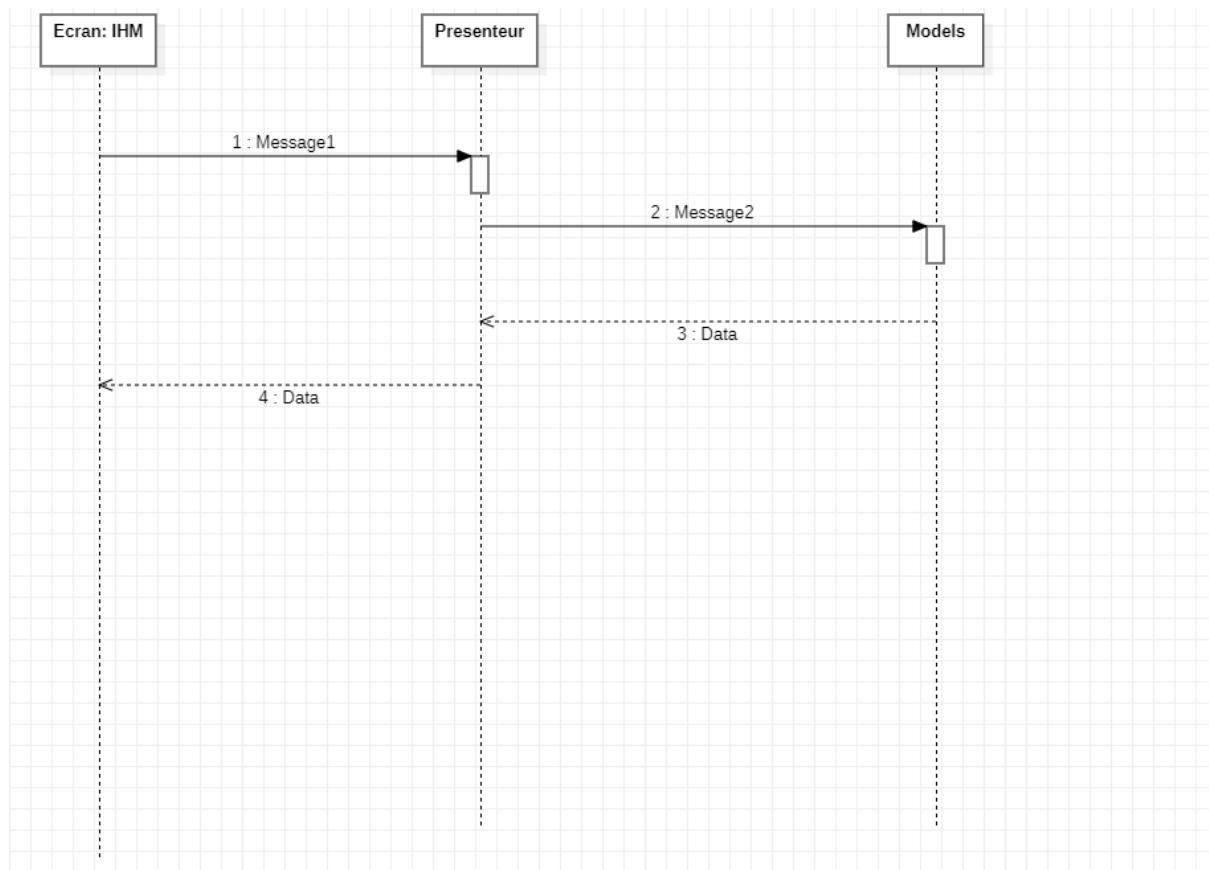
Les actions de l'utilisateur sont déportées dans les méthodes de la couche PRÉSENTATION, qui ensuite les redirigent éventuellement vers la couche Services .

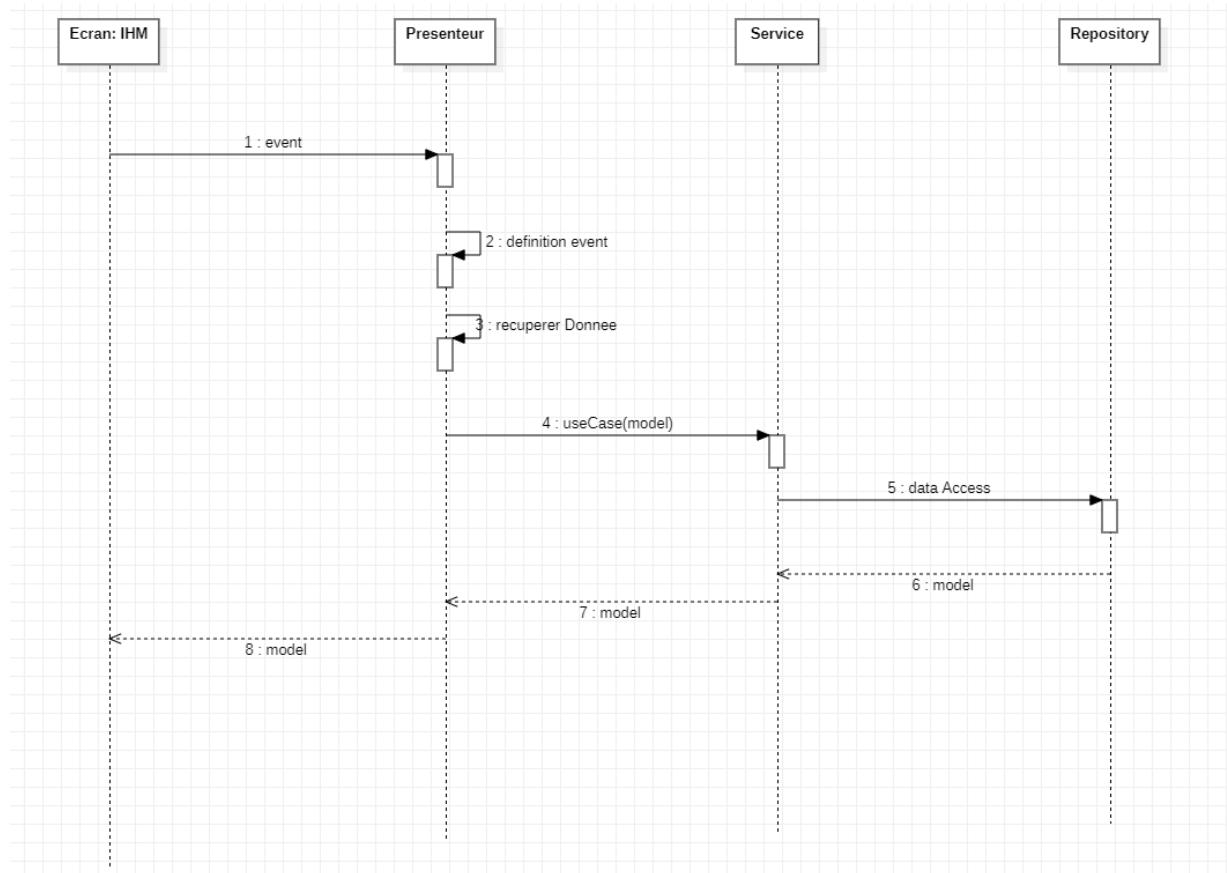
La couche PRESENTATION regroupe ainsi tous les traitements/événements concernant les actions de l'utilisateur. Le code est donc centralisé et partageable entre différentes VUES.

En revanche, la couche PRESENTATION n'accède pas à l'UI.

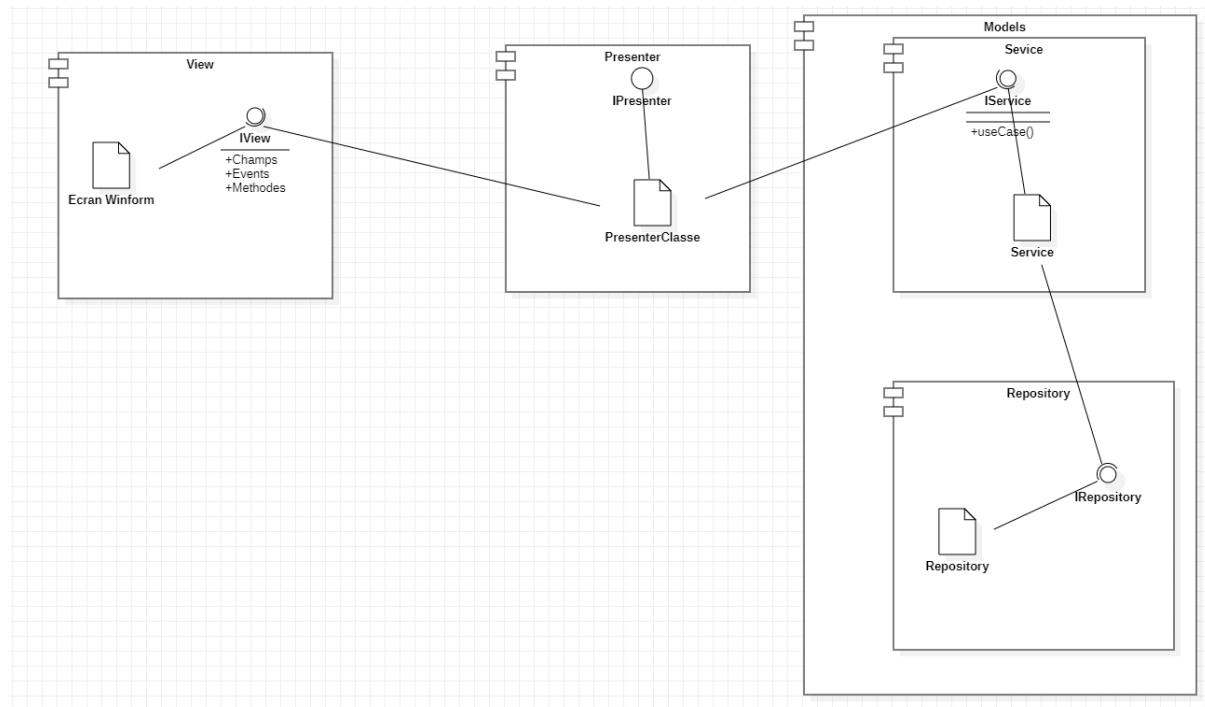
IV. Structuration du Projet

A. Diagramme de Séquence





B. Diagramme de Composant



Réalisation des Fonctionnalités

V. Création de la Structure du Projet

1. Créer un projet **WinForm**
2. Créer le Namespace **Models**
3. Créer le Namespace **views**
4. Créer le Namespace **Presenters**

NB:

En plus de ces trois couches, nous créerons les couches

5. Créer le Namespace **repositories**
6. Créer le Namespace **services**

VI. Gérer Classe

A. Couche Model

1.Création des Models

```
public class Grade
{
    0 références
    public int Id { get; set; }
    0 références
    public string Libelle { get; set; }
}

public class Classe
{
    private int id;
    private string libelle;
    private Grade grade;

    0 références
    public int Id { get; set; }
    0 références
    public string Libelle { get; set; }
    0 références
    public Grade Grade { get => grade; set => grade = value; }
}
```

```
0 références
public class Filiere
{
    private int id;
    private string libelle;

    0 références
    public int Id { get => id; set => id = value; }
    1 référence
    public string Libelle { get => libelle; set => libelle = value; }

    3 références
    public Filiere(int id, string libelle)
    {
        this.id = id;
        this.libelle = libelle;
    }

    0 références
    public Filiere()
    {
    }
}

16 références
public class Niveau
{
    private int id;
    private string libelle;

    0 références
    public int Id { get => id; set => id = value; }
    1 référence
    public string Libelle { get => libelle; set => libelle = value; }

    3 références
    public Niveau(int id, string libelle)
    {
        this.id = id;
        this.libelle = libelle;
    }

    0 références
    public Niveau()
    {
    }
}
```

2. Validation des Models

a) Ajouter la Référence

System.ComponentModel.DataAnnotations

b) Models Valider

```
2 références
public class Grade
{
    [System.ComponentModel.DisplayName("ID")]
    0 références
    public int Id { get; set; }
    [System.ComponentModel.DisplayName("Grade")]
    0 références
    public string Libelle { get; set; }
}

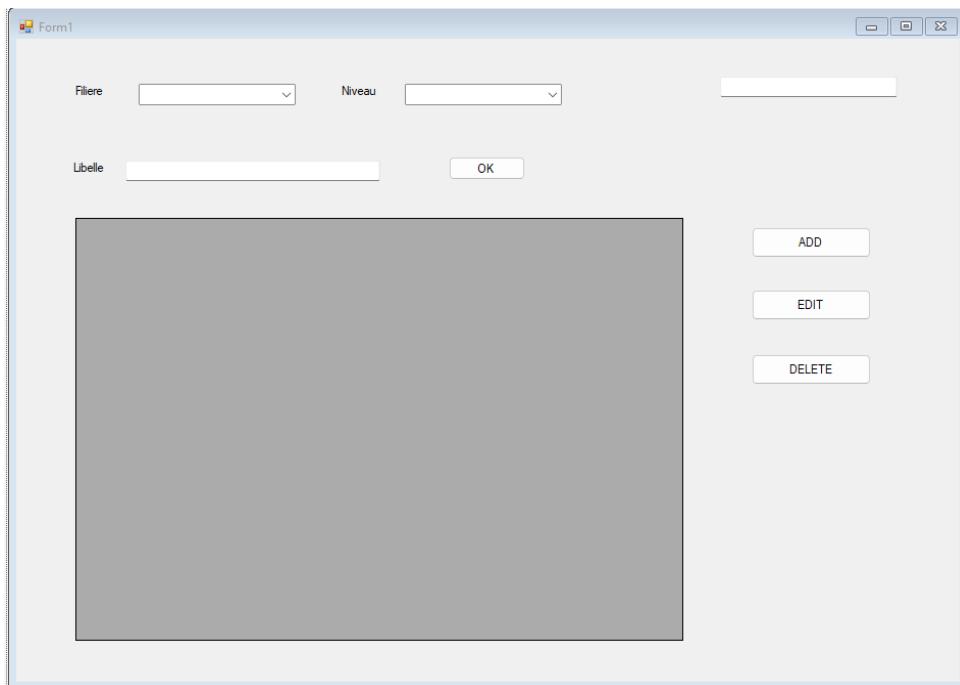
public class Classe
{
    private int id;
    private string libelle;
    private Grade grade;

    [System.ComponentModel.DisplayName("ID")]
    0 références
    public int Id { get; set; }

    [System.ComponentModel.DisplayName("Libelle")]
    [Required(ErrorMessage = "Libelle de classe est Obligatoire")]
    [StringLength(10, MinimumLength = 4, ErrorMessage = "Le Libelle de la classe doit avoir au minimum 4")]
    0 références
    public string Libelle { get; set; }
    0 références
    public Grade Grade { get => grade; set => grade = value; }
}
```

B. Couche View

#Interface Form1.cs[Desing]



#Interface IClasseView

```
4 références
public interface IViewClasse
{
    //Properties
    2 références
    int ClasseID { get; set; }
    3 références
    String LibelleRecherche { get; set; }
    2 références
    Filiere Filiere { get; set; }
    2 références
    Niveau Niveau { get; set; }
    1 référence
    String Message { get; set; }

    1 référence
    bool IsEdit { get; set; }
    1 référence
    bool IsSuccessfull { get; set; }

    //Events
    event EventHandler RechercherClasseEvent;
    event EventHandler AjouterClasseEvent;
    event EventHandler EditClasseEvent;
    event EventHandler SupprimerClasseEvent;
    event EventHandler AnnulerEvent;

    //Methods
    2 références
    void SetClasseBindingSource(BindingSource classeList, BindingSource filiereList, BindingSource NiveauList);
    0 références
    void Show();
}
```

#Ecran Form1.cs

```
public partial class Form1 : Form, IViewClasse
{
    private string message;
    private bool isSuccessfull;
    private bool isEdit;
    1 référence
    public Form1()
    {
        InitializeComponent();

        ActiveEventClasse();
    }

    //Redefinition des Propriétés
    2 références
    public int ClasseID { get => int.Parse(txtId.Text); set => txtId.Text = value.ToString(); }
    3 références
    public string LibelleRecherche { get => txtSearchLibelle.Text; set => txtSearchLibelle.Text = value; }
    2 références
    public Filiere Filiere { get => (Filiere)cboFiliere.SelectedItem; set => throw new NotImplementedException(); }
    2 références
    public Niveau Niveau { get => (Niveau)cboNiveau.SelectedItem; set => throw new NotImplementedException(); }
    1 référence
    public string Message { get => message; set => message = value; }
    1 référence
    public bool IsEdit { get => isSuccessfull; set => isSuccessfull = value; }
    1 référence
    public bool IsSuccessfull { get => isEdit; set => isEdit = value; }
```

```

//Redefinition des Evenements
public event EventHandler RechercherClasseEvent;
public event EventHandler AjouterClasseEvent;
public event EventHandler EditClasseEvent;
public event EventHandler SupprimerClasseEvent;
public event EventHandler AnnulerEvent;

//Activation des Evenements ou Definition des Ecouteurs Evenements
1 référence
private void ActiveEventClasse()
{
    btnSearch.Click += delegate { RechercherClasseEvent.Invoke(this, EventArgs.Empty); };
    txtSearchLibelle.KeyDown += (s, e) =>
    {
        if (e.KeyCode == Keys.Enter)
        {
            RechercherClasseEvent.Invoke(this, EventArgs.Empty);
        }
    };
    btnAdd.Click += delegate { AjouterClasseEvent.Invoke(this, EventArgs.Empty); };
    btnEdit.Click += delegate { EditClasseEvent.Invoke(this, EventArgs.Empty); };
    btnDel.Click += delegate { SupprimerClasseEvent.Invoke(this, EventArgs.Empty); };

    btnDel.Click += delegate { AnnulerEvent.Invoke(this, EventArgs.Empty); };
}

```

```

//Redefinition de la Methode SetClasseBindingSource pour le Data Binding
2 références
public void SetClasseBindingSource(BindingSource classeList, BindingSource filiereList,
                                    BindingSource NiveauList)
{
}

```

C. Réaliser Presenter

#ClassePresenter.cs

```

public class PresenterClasse:IPresenterClasse
{
    //Couplage Faible
    private IViewClasse view;
    private IService service;

    //Declaration des Bindings ou Propriete de Liaison de Donnees
    private BindingSource bindingSourceClasse=new BindingSource();
    private BindingSource bindingSourceFiliere = new BindingSource();
    private BindingSource bindingSourceNiveau = new BindingSource();

    // Declaration des Proprietes List pour l'initialisation des BindingSource
    private IEnumerable<Classe> classeList;
    private IEnumerable<Filiere> filiereList;
    private IEnumerable<Niveau> niveauList;

    //Injection de Dependence au niveau du Constructeur
    1 référence
    public PresenterClasse(IViewClasse view, IService service)
    {
        this.view = view;
        this.service = service;
    }
}

```

```

1 référence
public void BindingEventClasse()
{
    //Binding des Evenements de la vue
    this.view.RechercherClasseEvent += RechercherClasseHandler;
    this.view.AjouterClasseEvent += AjouterClasseHandler;
    this.view.EditClasseEvent += EditClasseHandler;
    this.view.SupprimerClasseEvent += SupprimerClasseHandler;
    this.view.AnnulerEvent += AnnulerActionHandler;
}

```

NB : La fonction BindingEventClasse sera appelée dans le constructeur

```

//Definition des Fonctions de Liaison D'evenements
1 référence
private void SupprimerClasseHandler(object sender, EventArgs e) { throw new NotImplementedException(); }
1 référence
private void AnnulerActionHandler(object sender, EventArgs e) { throw new NotImplementedException(); }
1 référence
private void EditClasseHandler(object sender, EventArgs e) { throw new NotImplementedException(); }
1 référence
private void AjouterClasseHandler(object sender, EventArgs e) { throw new NotImplementedException(); }
0 références
private void EditClasse(object sender, EventArgs e){ throw new NotImplementedException(); }
1 référence
private void RechercherClasseHandler(object sender, EventArgs e){ throw new NotImplementedException(); }
/*

```

D. Initialisation des Données au chargement de la Fenêtre

#FiliereRepository.cs

```

1 référence
public class FiliereRepository : IFiliereRepository
{
    2 références
    public List<Filiere> FindAll()
    {
        return new List<Filiere>()
        {
            new Filiere(1, "GLRS"),
            new Filiere(2, "MAE"),
            new Filiere(3, "ETSE"),
        };
    }
}

```

#NiveauRepository.cs

```
public class NiveauRepository : INiveauRepository
{
    2 références
    public List<Niveau> findAll()
    {

        return new List<Niveau>()
        {
            new Niveau(1, "L1"),
            new Niveau(2, "L2"),
            new Niveau(3, "L3"),
        };
    }
}
```

#Service.cs

```
2 références
public List<Filiere> listerFiliere()
{
    return filiereRepository.FindAll();
}

2 références
public List<Niveau> listerNiveau()
{
    return niveauRepository.findAll();
}
```

#Ecran Form1.cs

```
//Redefinition des Propriétés
1 référence
public Filiere Filiere { get => (Filiere)cboFiliere.SelectedItem; set => throw new NotImplementedException(); }
1 référence
public Niveau Niveau { get => (Niveau)cboNiveau.SelectedItem; set => throw new NotImplementedException(); }
1 référence
```

```

//Data Binding
2 références
public void SetClasseBindingSource(BindingSource classeList, BindingSource filiereList,
                                    BindingSource NiveauList)
{
    dtgClasse.DataSource = classeList;
    cboNiveau.DataSource = NiveauList;
    cboFiliere.DataSource = filiereList;

    cboNiveau.DisplayMember = "libelle";
    cboNiveau.ValueMember= "libelle";
    cboFiliere.DataSource = filiereList;
    cboFiliere.DisplayMember = "libelle";
    cboFiliere.ValueMember = "libelle";

}

```

#ClassePresenter.cs

```

//Fonction Initialisation des Donnes de la Vue Classe
0 références
private void initialize()
{
    filiereList=service.listerFiliere();
    niveauList=service.listerNiveau ();
    classeList=service.listeClasse();
    bindingSourceFiliere.DataSource=filiereList;
    bindingSourceNiveau.DataSource=niveauList;
    bindingSourceClasse.DataSource=classeList;
    this.view.SetClasseBindingSource(bindingSourceClasse, bindingSourceFiliere, bindingSourceNiveau);
}

```

NB : La fonction BindingEventClasse sera appelée dans le constructeur

#Program.cs => inversion de contrôle et Création des Dépendances

```

Application.SetCompatibleTextRenderingDefault(false);
IViewClasse view = new Form1();
IFiliereRepository filiereRepository=new FiliereRepository();
INiveauRepository niveauRepository=new NiveauRepository();
IClasseRepository classeRepository=new ClasseRepository();
IService service = new Service(filiereRepository, niveauRepository, classeRepository);
IPresenterClasse presenter = new PresenterClasse(view,service);
Application.Run((Form)view);

```

ADO.NET

Accès aux bases de données avec C#

ADO.NET : Accès aux bases de données avec C#

- Mode déconnecté
- DataSet, DataTable, DataAdapter
- Gestion des transactions
- Applications WindowsForms
- Utilisation des assistants Visual Studio

ADO.NET : Accès aux bases de données avec C#

❖ Présentation

□ ADO.NET est un ensemble de classes qui exposent les services d'accès aux données pour les programmeurs .NET Framework. ADO.NET propose un large ensemble de composants pour la création d'applications distribuées avec partage de données.

Partie intégrante du .NET Framework, il permet d'accéder à des données:

- relationnelles,
 - XML
 - d'application. ADO.NET
- ADO.NET répond à divers besoins en matière de développement, en permettant notamment:
- de créer des clients de bases de données frontaux
 - des objets métier de couche intermédiaire utilisés par
 - des applications,
 - outils, langages
 - ou navigateurs Internet.

□ ADO.NET sépare l'accès aux données de leur manipulation en composants distincts qui peuvent être utilisés individuellement ou en tandem. ADO.NET comprend des fournisseurs de données .NET Framework pour la connexion à une base de données, l'exécution de commandes et l'extraction de résultats. Fournisseurs de données .NET Framework les utilisés sont:

- **System.Data.SqlClient** Pour les sources de données SQL SERVER
- **System.Data.OleDb** Pour les sources de données exposées à l'aide de OLE DB.
- **System.Data.Odbc** Pour les sources de données exposées à l'aide d' ODBC.
- Pour les sources de données Oracle et utilise l'espace de noms **System.Data.OracleClient**.
- Pour les sources de données Mysql et utilise l'espace de noms **System.Data.MySqlConnection**.
- Fournit un accès aux données pour les applications EDM (Entity Data Model) et utilise l'espace de noms **System.Data.EntityClient**.

□ Objets principaux des fournisseurs de données .NET Framework

object	Description
Connection	Établit une connexion à une source de données spécifique. La classe de base pour tous les objets Connection est la classe DbConnection .
Command	Exécute une commande sur une source de données. Expose Parameters et peut exécuter dans la portée d'une Transaction à partir d'un Connection . La classe de base pour tous les objets Command est la classe DbCommand .
DataReader	Lit un flux de données avant uniquement (<i>forward only</i>) et en lecture seule à partir d'une source de données. La classe de base pour tous les objets DataReader est la classe DbDataReader .
DataAdapter	Remplit un DataSet et répercute les mises à jour dans la source de données. La classe de base pour tous les objets DataAdapter est la classe DbDataAdapter .

Un objet **Connection** est utilisé conjointement à la classe

DataAdapter .

NB : Dans ce cours nous utiliserons le SGBD SQLSERVER donc :

- L'espace de Nom sera **System.Data.SqlClient**
- L' objet **Connection** va correspondre à **SqlConnection**
- L' objet **Command** va correspondre à **SqlCommand**
- L' objet **DataAdapter** va correspondre à **SqlDataAdapter**
- L' objet **DataReader** va correspondre à **SqlDataReader**

1. Méthode d'utilisation de ADO.NET en Mode Connecté

a. Importation des Namespace:

- **using System.Data;**
- **using System.Data.SqlClient;**

b. Etablir une Connection à la Source de Donnée

i. Récupérer la Chaîne de Connexion

```
String ChaineConnection = «  
    "Data Source= serveur \SQLEXPRESS; => Instance  
    Initial Catalog= NomBaseDonnee; => Base de Donnée  
    Integrated Security= True " => Type  
    d'authentification par défaut Windows
```

ii. Créer un objet de Type SqlConnection

```
SqlConnection Conn = new SqlConnection( ChaineConnection);
```

iii. C) Ouvrir la Connexion

```
Conn.open();
```

c. Extraire les données de la source de données.

i. Ecrire la requête avec

- Requête non paramétrée avec **string.Format**
- Requête paramétrée et le paramètre est représenté par **@nomParametre**

ii. Créer Un Objet SqlCommand

```
SqlCommand cmd = new SqlCommand("requete", Conn);
```

NB : Si la requête est paramétrée, il faut remplacer les paramètres par leurs valeurs dans la requête. On créera un Objet de la Classe **SqlParameter** comme suit:

```
SqlParameter param=new SqlParameter("@nomParametre", valeur);
```

Ajouter le Paramètre **cmd.Parameters.Add(param);**

- iii. Exécution de la requête**
 - Requête Select: **SqlDataReader dr = cmd.ExecuteReader();**
 - Requête mise à jour : int **cmd.ExecuteNonQuery();**
 - iv. Parcourir le SqlDataReader avec la méthode dr.Read()**
 - v. A la fin fermer le SqlDataReader avec dr.Close().**
- d. Fermer la Connection avec Conn.close();**

E. Lister Classe avec Filtrer

#BaseRepository.cs

```
1 référence
public class BaseRepository
{
    private string connectionString;
    3 références
    public string ConnectionString { get => connectionString; set => connectionString = value; }
}
```

#ClasseRepository.cs

```
1 référence
public ClasseRepository(string connectionString)
{
    ConnectionString= connectionString;
}

2 références
public List<Classe> FindAll()
{
    List<Classe> classeList = new List<Classe>();
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        try
        {
            connection.Open();
            cmd.Connection = connection;
            cmd.CommandText = @"Select * from classe";

            using (var reader = cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    var classe = new Classe()
                    {
                        Id = (int)reader[0],
                        Libelle = reader[1].ToString(),
                    };

                    classeList.Add(classe);
                }
            }
        }
        catch (SqlException ex)
        {
        }
    }
    return classeList;
}
```

```

2 références
public List<Classe> FindByLibelle(string Libelle)
{
    List<Classe> classeList = new List<Classe>();
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        connection.Open();
        cmd.Connection= connection;
        cmd.CommandText = @"Select * from classe where libelle like '%'+'@libelle+'%'";
        cmd.Parameters.AddWithValue("@libelle", SqlDbType.NVarChar).Value=Libelle ;
        using (var reader = cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                var classe = new Classe()
                {
                    Id = (int)reader[0],
                    Libelle = reader[1].ToString(),
                };

                classeList.Add((classe));
            }
        }
    }
    return classeList;
}

```

#Program.cs

```

Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
IViewClasse view = new Form1();
IFiliereRepository filiereRepository=new FiliereRepository();
INiveauRepository niveauRepository=new NiveauRepository();
string connectionString = @"Data Source=FATOUWADE\SQLEXPRESS;Initial Catalog=demon;Integrated Security=True";

IClasseRepository classeRepository=new ClasseRepository(connectionString);

IService service = new Service(filiereRepository, niveauRepository,classeRepository);
IPresenterClasse presenter = new PresenterClasse(view,service);
Application.Run((Form)view);

```

NB : La chaîne de Connexion peut être défini dans le fichier App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="demon.Properties.Settings.demonConnectionString"
            connectionString="Data Source=FATOUWADE\SQLEXPRESS;Initial Catalog=demon;Integrated Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
    </startup>
</configuration>

```

#Program.cs

Ajouter le NameSpace : **System.Configuration.ConfigurationManager**

```
string connectionString = ConfigurationManager.ConnectionStrings["GES_ETU"].ConnectionString;
IClasseRepository classeRepository=new ClasseRepository(connectionString);
```

F. Ajouter une Classe

#IClasseRepository.cs

2 références

```
void Add(Classe classe);
```

#ClasseRepository.cs

2 références

```
public void Add(Classe classe)
{
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        try
        {
            connection.Open();
            cmd.Connection = connection;
            cmd.CommandText = @"insert into classes(libelle,filiere,niveau) values(@libelle,@filiere,@niveau)";
            cmd.Parameters.AddWithValue("@libelle", SqlDbType.NVarChar).Value = classe.Libelle;
            cmd.Parameters.AddWithValue("@filiere", SqlDbType.NVarChar).Value = classe.Filiere;
            cmd.Parameters.AddWithValue("@niveau", SqlDbType.NVarChar).Value = classe.Niveau;

            cmd.ExecuteNonQuery();
        }
        catch (SqlException ex)
        {
        }
        finally
        {
            connection.Close();
        }
    }
}
```

#IClasseService.cs

2 références

```
void AjouterClasse(Filiere filiere, Niveau niveau);
```

2 références

#ClasseService.cs

2 références

```
public void AjouterClasse(Filiere filiere, Niveau niveau)
{
    string libelle = String.Format("{0}{1}", niveau.Libelle,filiere.Libelle);
    classeRepository.Add(new Classe()
    {
        Libelle = libelle,
        Niveau= niveau.Libelle,
        Filiere=filiere.Libelle
    });
}
```

#Ecran Form1.cs

```
btnAdd.Click += delegate {
    AjouterClasseEvent.Invoke(this, EventArgs.Empty);
    MessageBox.Show(message, "Gestion Etudiant", MessageBoxButtons.OK, MessageBoxIcon.Information);
};
```

#ClassePresenter.cs

```
//Refresh le DataGridView après insertion
4 références
private void loadDataClasse()
{
    classeList = service.listerClasse();
    bindingSourceClasse.DataSource = classeList;
}

private void AjouterClasseHandler(object sender, EventArgs e) {

    try
    {
        view.IsEdit = false;
        Filiere filiere = bindingSourceFiliere.Current as Filiere;
        Niveau niveau = bindingSourceNiveau.Current as Niveau;
        service.AjouterClasse(filiere, niveau);
        view.Message = "classe ajoute avec succès";
        loadDataClasse();
    }
    catch (Exception ex)
    {
        view.Message = ex.Message;
    }
}
```

G. Editer une Classe

#IviewClasse.cs

```
EVENT EVENTADDEVENT DELETEDCLASSEVENT,
```

```
event EventHandler SeleledClasseEvent;
```

#Ecran Form1.cs

```
dtgClasse.SelectionChanged += delegate { SeleledClasseEvent.Invoke(this, EventArgs.Empty); };

3 références
public Filiere Filiere { get => (Filiere)cboFiliere.SelectedItem; set => cboFiliere.Text = value.Libelle; }
3 références
public Niveau Niveau { get => (Niveau)cboNiveau.SelectedItem; set => cboNiveau.Text = value.Libelle; }
5 références
```

#ClassePresenter.cs

```
this.view.SeleledClasseEvent += SeleledClasseHandler;
```

```
1 référence
private void SeleledClasseHandler(object sender, EventArgs e)
{
    classeSelect = (Classe)bindingSourceClasse.Current;

    view.Filiere = new Filiere() { Libelle = classeSelect.Filiere };
    view.Niveau = new Niveau() { Libelle = classeSelect.Niveau };
    view.IsEdit = true;
}
```

H. Modifier une Classe

#IClasseRepository.cs

```
2 références
void Update(Classe classe);
```

2 références

#ClasseRepository.cs

```
2 références
public void Update(Classe classe)
{
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        try
        {
            connection.Open();
            cmd.Connection = connection;
            cmd.CommandText = @"update classes set libelle=@libelle,filiere=@filiere,niveau=@niveau where id=@id";
            cmd.Parameters.Add("@libelle", SqlDbType.NVarChar).Value = classe.Libelle;
            cmd.Parameters.Add("@filiere", SqlDbType.NVarChar).Value = classe.Filiere;
            cmd.Parameters.Add("@niveau", SqlDbType.NVarChar).Value = classe.Niveau;
            cmd.Parameters.Add("@id", SqlDbType.Int).Value = classe.Id;
            cmd.ExecuteNonQuery();

        }
        catch (SqlException ex)
        {

        }
        finally
        {
            connection.Close();
        }
    }
}
```

#IClasseService.cs

2 références

```
void ModifierClasse(Classe classe);
```

#ClasseService.cs

```
2 références
public void ModifierClasse(Classe classe)
{
    string libelle = String.Format("{0}{1}", classe.Niveau, classe.Filiere);
    classe.Libelle = libelle;
    classeRepository.Update(classe);
}
```

#Ecran Form1.cs

```
btnEdit.Click += delegate {
    EditClasseEvent.Invoke(this, EventArgs.Empty);
    MessageBox.Show(message, "Gestion Etudiant", MessageBoxButtons.OK, MessageBoxIcon.Information);
};
```

#ClassePresenter.cs

```
this.view.EditClasseEvent += EditClasseHandler;

private void EditClasseHandler(object sender, EventArgs e)
{

    classeSelect.Filiere = view.Filiere.Libelle;
    classeSelect.Niveau = view.Niveau.Libelle;
    service.ModifierClasse(classeSelect);
    view.Message = "classe Modifiee avec succes";
    loadDataClasse();
}
--
```

I. Supprimer une Classe

#IClasseRepository.cs

```
2 références
void Delete(int id);
-----
```

#ClasseRepository.cs

2 références

```
public void Delete(int id)
{
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        try
        {
            connection.Open();
            cmd.Connection = connection;
            cmd.CommandText = @"delete from classes where id=@id";
            cmd.Parameters.AddWithValue("@id", SqlDbType.Int).Value = id;
            cmd.ExecuteNonQuery();

        }
        catch (SqlException ex)
        {
        }
        finally
        {
            connection.Close();
        }
    }
}
```

#IClasseService.cs

2 références

```
void SupprimerClasse(int id);
```

#ClasseService.cs

✓ references

```
public void SupprimerClasse(int id)
{
    classeRepository.Delete(id);
}
```

#Ecran Form1.cs

```
btnDel.Click += delegate {

    var result= MessageBox.Show("Veuillez confirmer la suppression ?","Attention",
        MessageBoxButtons.YesNo,MessageBoxIcon.Warning);
    if (result == DialogResult.Yes)
    {
        SupprimerClasseEvent.Invoke(this, EventArgs.Empty);
        MessageBox.Show(message, "Gestion Etudiant", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

};
```

#ClassePresenter.cs

```
this.view.SupprimerClasseEvent += SupprimerClasseHandler;

private void SupprimerClasseHandler(object sender, EventArgs e) {
    service.SupprimerClasse(classeSelect.Id);
    view.Message = "classe Supprimer avec succes";
    loadDataClasse();
}

}
```

J. Validation du Modèle

a) Ajouter la Référence

System.ComponentModel.DataAnnotations

b) Les Validators

Required

System.ComponentModel.DataAnnotations Attributes

Attribute	Description
Key	Can be applied to a property to specify a key property in an entity and make the corresponding column a PrimaryKey column in the database.
Timestamp	Can be applied to a property to specify the data type of a corresponding column in the database as <code>rowversion</code> .
ConcurrencyCheck	Can be applied to a property to specify that the corresponding column should be included in the optimistic concurrency check.
Required	Can be applied to a property to specify that the corresponding column is a NotNull column in the database.
MinLength	Can be applied to a property to specify the minimum string length allowed in the corresponding column in the database.
MaxLength	Can be applied to a property to specify the maximum string length allowed in the corresponding column in the database.
StringLength	Can be applied to a property to specify the maximum string length allowed in the corresponding column in the database.

Exemple

```
[System.ComponentModel.DisplayName("Niveau")]
[Required(ErrorMessage = "Veuillez Selectionner au moins un Niveau")]

```

```
Range(0 , double.MaxValue, ErrorMessage = "The value must be greater than 0")]
```

2. Classe de Validation

```
1 référence
public class ModelDataValidation
{
    1 référence
    public void validate(object model)
    {
        String errorMessage = "";
        List<ValidationResult> results = new List<ValidationResult>();
        ValidationContext context = new ValidationContext(model);
        bool isValid = Validator.TryValidateObject(model, context, results, true);
        if (!isValid)
        {
            foreach (ValidationResult validationResult in results)
            {
                errorMessage += " - " + validationResult.ErrorMessage + "\n";
                throw new Exception(errorMessage);
            }
        }
    }
}
```

3. Crédation d'une fonction de validation

```
2 références
public class FiliereValidate : ValidationAttribute
{
    1 référence
    public FiliereValidate() : base("{0} est Obligatoire")
    {
    }

    0 références
    protected override ValidationResult IsValid(object value, ValidationContext validationContext)
    {
        Filiere filiere = value as Filiere;
        bool valid = filiere == null;
        if (valid)
        {
            return new ValidationResult(base.FormatErrorMessage(validationContext.MemberName)
                , new string[] { validationContext.MemberName });
        }

        return null;
    }
}
```

4. Validation de l'entité classe

```
[System.ComponentModel.DisplayName("Libelle")]
7 références
public string Libelle { get => libelle; set => libelle = value; }

[System.ComponentModel.DisplayName("Filiere")]
[FiliereValidate]
9 références
public string Filiere { get => filiere; set => filiere = value; }

[System.ComponentModel.DisplayName("Niveau")]
9 références
public string Niveau { get => niveau; set => niveau = value; }
```

5. Ajout d'une classe avec Validation

#IClasseService.cs

```
2 références
void AjouterClasse(Classe classe);
2 références
```

#ClasseService.cs

```
↳ references
public void AjouterClasse(Classe classe)
{
    classeRepository.Add(classe);
}
```

#ClassePresenter.cs

```
private void AjouterClasseHandler(object sender, EventArgs e) {

    try
    {
        view.IsEdit = false;
        Filiere filiere = bindingSourceFiliere.Current as Filiere;
        Niveau niveau = bindingSourceNiveau.Current as Niveau;

        var model= new Classe()
        {
            Libelle = String.Format("{0}{1}", niveau.Libelle, filiere.L
            Niveau = niveau.Libelle,
            Filiere = filiere.Libelle
        };
        new ModelDataValidation().validate(model);

        service.AjouterClasse(model);
        view.Message = "classe ajoute avec succes";
        loadDataClasse();
        this.view.IsSuccessfull = true;
    }
    catch (Exception ex)
    {
        this.view.IsSuccessfull=false;
        view.Message = ex.Message;
    }

}
```

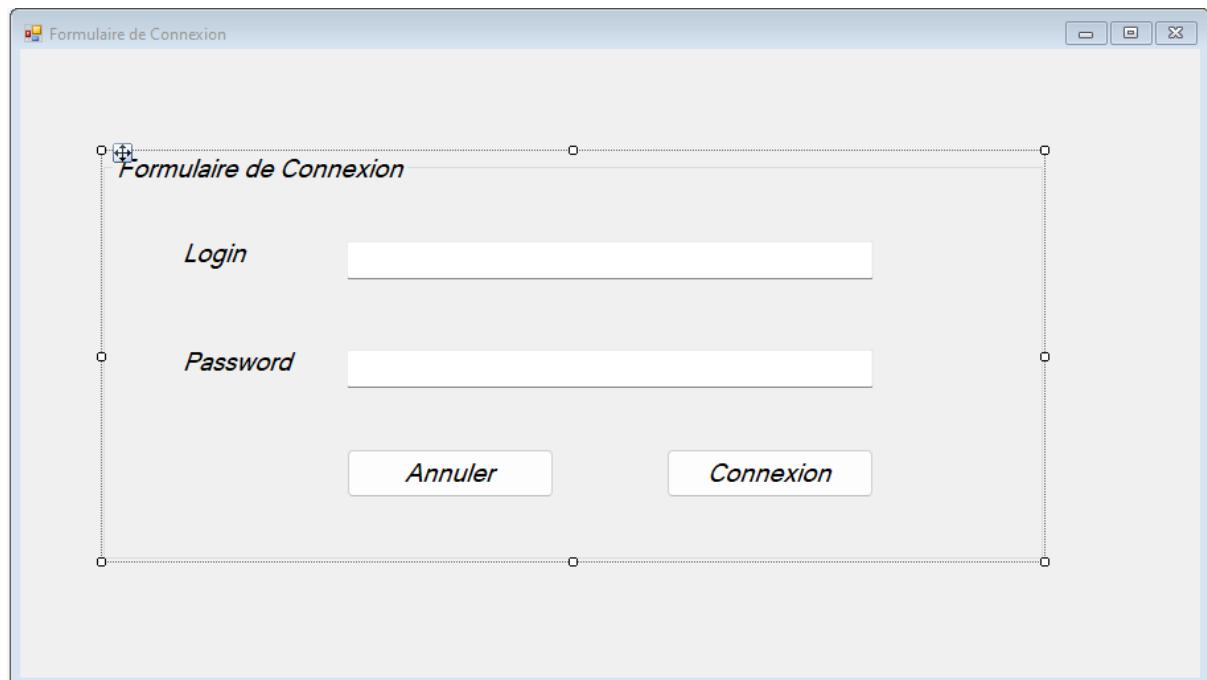
K. Mettre en place la classe Fabrique

```
3 références
public class Fabrique
{
    private static IService instance;
    3 références
    public static IService GetInstance()
    {
        if(instance == null)
        {
            string connectionString = ConfigurationManager.ConnectionStrings["GES_ETU"].ConnectionString;
            instance = new Service(
                new FiliereRepository(),
                new NiveauRepository(),
                new ClasseRepository(connectionString),
                new UserRepository(connectionString)
            );
        }

        return instance;
    }
}
```

L. Gestion de la connexion

#Ecran de Connexion



#Models>User+Validation

```
public enum Role { ROLE_AC, ROLE_RP, ROLE_ETUDIANT }
10 références
public class User
{
    private int id;
    private string nomComplet;
    private string login;
    private string password;
    Role role;

    [System.ComponentModel.DisplayName("ID")]
    1 référence
    public int Id { get => id; set => id = value; }
    1 référence
    public string NomComplet { get => nomComplet; set => nomComplet = value; }

    [System.ComponentModel.DisplayName("Login")]
    [Required(ErrorMessage ="Le Login est Obligatoire")]

    2 références
    public string Login { get => login; set => login = value; }
    [System.ComponentModel.DisplayName("Password")]
    [Required(ErrorMessage = "Le Mot de Passe est Obligatoire")]
    [StringLength(10,MinimumLength =2, ErrorMessage = "Le Mot de Passe doit contenir au moins 2 caractere")]
    3 références
    public string Password { get => password; set => password = value; }
    1 référence
    public Role RoleUser { get => role; set => role = value; }

}
```

#IUserRepository

```
public interface IUser
{
    3 références
    User FindUserByLogin(string login);
}
```

#UserRepository

```

public class UserRepository :BaseRepository, IUser
{
    1 référence
    public UserRepository(String connectionString)
    {
        this.ConnectionString = connectionString;
    }

    3 références
    public User FindUserByLogin(string login)
    {
        User user = null;
        using (var connection = new SqlConnection(ConnectionString))
        using (var cmd = new SqlCommand())
        {
            connection.Open();
            cmd.Connection = connection;
            cmd.CommandText = @"Select * from users where login = @login";
            cmd.Parameters.Add("@login", SqlDbType.NVarChar).Value = login;

            using (var reader = cmd.ExecuteReader())
            {
                if (reader.Read())
                {
                    user = new User()
                    {
                        Id = int.Parse(reader[0].ToString()),
                        Login = reader[1].ToString(),
                        NomComplet = reader[2].ToString(),
                        Password = reader[3].ToString(),
                    };
                    Enum.TryParse(reader[4].ToString(), out Role role);
                    user.RoleUser = role;
                }
            }
        }
        return user;
    }
}

```

#IService

```

bool UserExist(string login);
2 références
User seConnecter(string login, string password);

```

#Service

```

2 références
public bool UserExist(string login)
{
    return userRepository.FindUserByLogin(login) != null;
}
2 références
public User seConnecter(string login, string password)
{
    User user = userRepository.FindUserByLogin(login);
    return (user == null || user.Password != password) ? null : user;
}

```

#Views>IViewConnexion

```

4 références
public interface IViewConnexion
{
    3 références
    string Login { get; set; }
    3 références
    string Password { get; set; }
    4 références
    string Message { get; set; }
    4 références
    bool IsSucess { get; set; }

    event EventHandler ConnexionEvent;
    event EventHandler AnnulerEvent;

    0 références
    void Show();
    1 référence
    void Hide();
}

```

#Views>FrmConexion

```

public partial class FrmConnexion : Form, IViewConnexion
{
    1 référence
    public FrmConnexion()
    {
        InitializeComponent();
        btnConnexion.Click += delegate {

            ConnexionEvent.Invoke(this, EventArgs.Empty);
            if(!IsSucess) MessageBox.Show(Message);

        };
        btnAnnuler.Click += delegate { AnnulerEvent.Invoke(this, EventArgs.Empty); };
    }

    3 références
    public string Login { get => txtLogin.Text.Trim(); set => txtLogin.Text = value; }
    3 références
    public string Password { get => txtPassword.Text.Trim(); set => txtPassword.Text = value; }
    4 références
    public string Message { get ; set ; }
    4 références
    public bool IsSucess { get ; set; }

    public event EventHandler ConnexionEvent;
    public event EventHandler AnnulerEvent;

    1 référence
    private void FrmConnexion_Load(object sender, EventArgs e)
    {
    }
}

```

IConnexionPresenter

ConnexionPresenter

```
public class PresenterConnexion:IPresenterConnexion
{
    IViewConnexion view;
    IService service;

    1 référence
    public PresenterConnexion(IViewConnexion view, IService service)
    {
        this.view = view;
        this.service = service;
        this.view.ConnexionEvent += ConnexionHandler;
    }

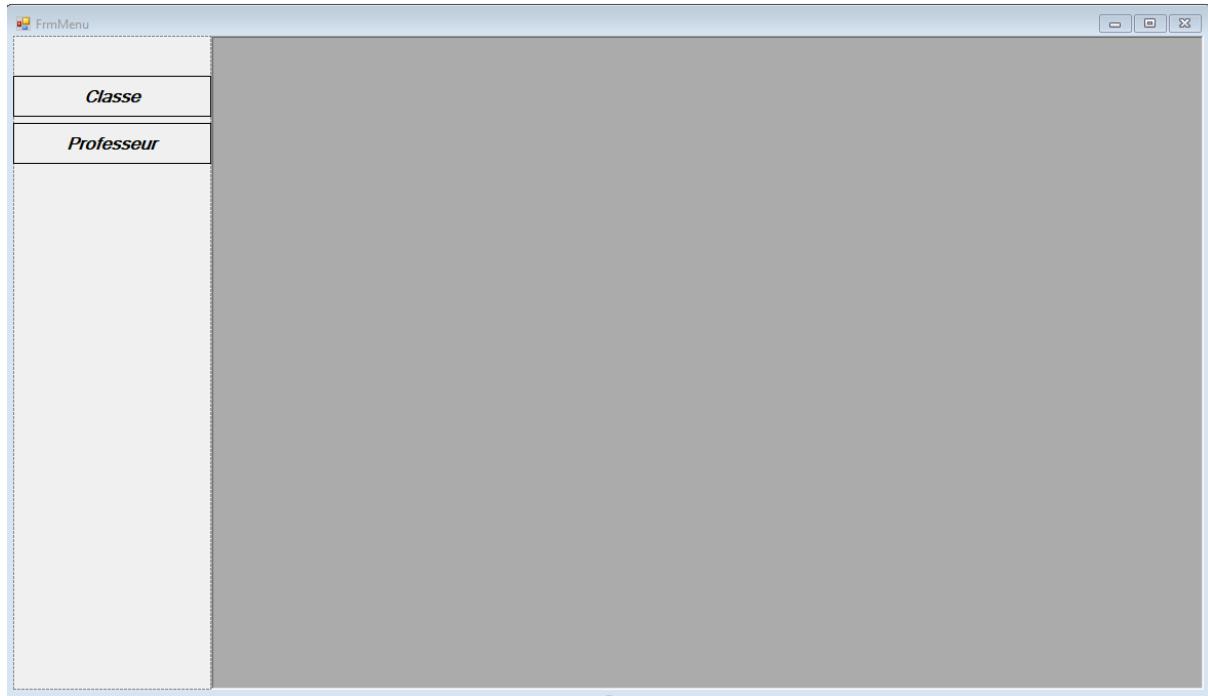
    1 référence
    private void ConnexionHandler(object sender, EventArgs e)
    {
        this.view.IsSuccess = false;
        try
        {
            User user = new User() { Login = this.view.Login, Password = this.view.Password };
            new ModelDataValidation().validate(user);
            user=service.seConnecter(this.view.Login, this.view.Password);
            if (user != null)
            {
                IViewMenu viewMenu = new FrmMenu();
                IPresenterMenu presenterMenu =new PresenterMenu(viewMenu);
                view.Hide();
                this.view.IsSuccess = true;
            }
            else
            {
                this.view.Message = "Login ou Mot de Passe Incorrect";
            }
        }
        catch (Exception ex)
        {
            this.view.Message=ex.Message;
        }
    }
}
```

#Program.cs

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    IViewConnexion view = new FrmConnexion();
    // string connectionString = @"Data Source=FATOUWADE\SQLEXPRESS;Initial Catalog=demon;Integrated Security=True";
    IPresenterConnexion presenter = new PresenterConnexion(view, Fabrique.GetInstance());
    Application.Run((Form)view);
}
```

M. Mettre en Place le Menu

#FrmMenu



#IViewMenu

```
public interface IViewMenu
{
    event EventHandler showClasseViewEvent;
    event EventHandler showProfesseurViewEvent;
    1 référence
    void Show();
}
```

#FrmMenu.cs

```
public partial class FrmMenu : Form, IViewMenu
{
    1 référence
    public FrmMenu()
    {
        InitializeComponent();
        btnClasse.Click += delegate { showClasseViewEvent.Invoke(this, EventArgs.Empty); };
        btnProfesseur.Click+= delegate { showProfesseurViewEvent.Invoke(this, EventArgs.Empty); };
    }

    public event EventHandler showClasseViewEvent;
    public event EventHandler showProfesseurViewEvent;
}
```

NB : Gestion de la Fenêtre MDI

#FrmClasse.cs

```

private static Form1 instance = null;
1 référence
public static Form1 getInstance(Form parentForm)
{
    if (parentForm.ActiveMdiChild != null)
    {
        parentForm.ActiveMdiChild.Close();
    }
    if (instance == null || instance.IsDisposed)
    {
        instance = new Form1();
        instance.FormBorderStyle= FormBorderStyle.None;
        instance.MdiParent=parentForm;
    }
    else
    {
        if(instance.WindowState == FormWindowState.Minimized)
        {
            instance.WindowState = FormWindowState.Normal;
        }
        instance.BringToFront();
    }
    return instance;
}

```

#FrmProfesseur.cs

```

private static FrmProfesseur instance = null;
1 référence
public static FrmProfesseur getInstance(Form parentForm)
{
    if(parentForm.ActiveMdiChild != null)
    {
        parentForm.ActiveMdiChild.Close();
    }
    if (instance == null || instance.IsDisposed)
    {
        instance = new FrmProfesseur();
        instance.FormBorderStyle = FormBorderStyle.None;
        instance.MdiParent = parentForm;
    }
    else
    {
        if (instance.WindowState == FormWindowState.Minimized)
        {
            instance.WindowState = FormWindowState.Normal;
        }
        instance.BringToFront();
    }
    return instance;
}

```

NB : Fermer la fenêtre en cours avant d'ouvrir une autre fenêtre

```
if (parentForm.ActiveMdiChild != null)
{
    parentForm.ActiveMdiChild.Close();
}
```

#PresenterMenu

```
public class PresenterMenu:IPresenterMenu
{
    private IViewMenu viewMenu;
    private IService service;

    1 référence
    public PresenterMenu(IViewMenu viewMenu )
    {
        this.viewMenu = viewMenu;
        service=Fabrique.GetInstance();

        this.viewMenu.showClasseViewEvent += showClasseViewHandler;
        this.viewMenu.showProfesseurViewEvent += showProfViewHandler;
        this.viewMenu.Show();
        loadViewClasse();
    }

    1 référence
    private void showProfViewHandler(object sender, EventArgs e)
    {
        IViewProf viewProf = FrmProfesseur.getInstance((Form)viewMenu);
        new PresenterProf(viewProf, service);
    }

    1 référence
    private void showClasseViewHandler(object sender, EventArgs e) { loadViewClasse();}

    2 références
    private void loadViewClasse()
    {
        IViewClasse viewClasse = Form1.getInstance((Form)viewMenu);
        new PresenterClasse(viewClasse, service);
    }
}
```

N. Gestion des Annees Scolaires

1. Back end

#Repository>AnneeRepository.cs

2 références

List<AnneeScolaire> FindAll();

2 références

AnneeScolaire FindByEtat(int etat);

```
public AnneeRepository(string connectionString)
{
    ConnectionString = connectionString;
}
```

```
public List<AnneeScolaire> FindAll()
{
    List<AnneeScolaire> annees = new List<AnneeScolaire>();
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        try
        {
            connection.Open();
            cmd.Connection = connection;
            cmd.CommandText = @"Select * from anneescolaire order by etat desc";

            using (var reader = cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    var annee = new AnneeScolaire()
                    {
                        Id = (int)reader[0],
                        Libelle = reader[1].ToString(),
                        Etat= (int)reader[2]
                    };

                    annees.Add((annee));
                }
            }
        }
        catch (SqlException ex)
        {
            throw new Exception(ex.Message);
        }
        finally
        {
            connection.Close();
        }
    }
    return annees;
}
```

```

2 références
public AnneeScolaire FindByEtat(int etat)
{
    AnneeScolaire annee = null;
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        connection.Open();
        cmd.Connection = connection;
        cmd.CommandText = @"Select * from anneescolaire where etat=@etat";
        cmd.Parameters.Add("@etat", SqlDbType.Int).Value = etat;
        using (var reader = cmd.ExecuteReader())
        {
            if (reader.Read())
            {
                annee = new AnneeScolaire()
                {
                    Id = (int)reader[0],
                    Libelle = reader[1].ToString(),
                    Etat =(int) reader[2]
                };
            }
        }
    }
    return annee;
}

```

#Service>Service.cs

```

2 references
List<AnneeScolaire> listerAnnee();
2 références
AnneeScolaire listerAnnee(int etat);

2 références
public List<AnneeScolaire> listerAnnee()
{
    return anneRepository.FindAll();
}

2 références
public AnneeScolaire listerAnnee(int etat)
{
    return anneRepository.FindByEtat(etat);
}

```

2. Front end

a) Récupérer l'année En Cours après connexion

#views>IViewMenu.cs

```

4 références
AnneeScolaire AnneeEncours { get; set; }
2 références

```

#views>FormMenu.cs

```
public string AnneeLabel { get => lblAnnee.Text; set => lblAnnee.Text = value; }
```

#Presenter>PresenterConnexion.cs>ConnexionHandler

```
if (user != null)
{
    IViewMenu viewMenu = new FrmMenu();
    viewMenu.UserConnect = user;
    viewMenu.AnneeEncours = service.listerAnnee(1);
    IPresenterMenu presenterMenu = new PresenterMenu(viewMenu);
    view.Hide();
    this.view.IsSuccess = true;
}
else
{
    this.view.Message = "Login ou Mot de Passe Incorrect";
}
```

#Presenter>PresenterMenu.cs>Constructeur

```
this.viewmenu.userLabel = [lab] pour accepter ↗
this.viewMenu.AnneeLabel = this.viewMenu.AnneeEncours.Libelle;
```

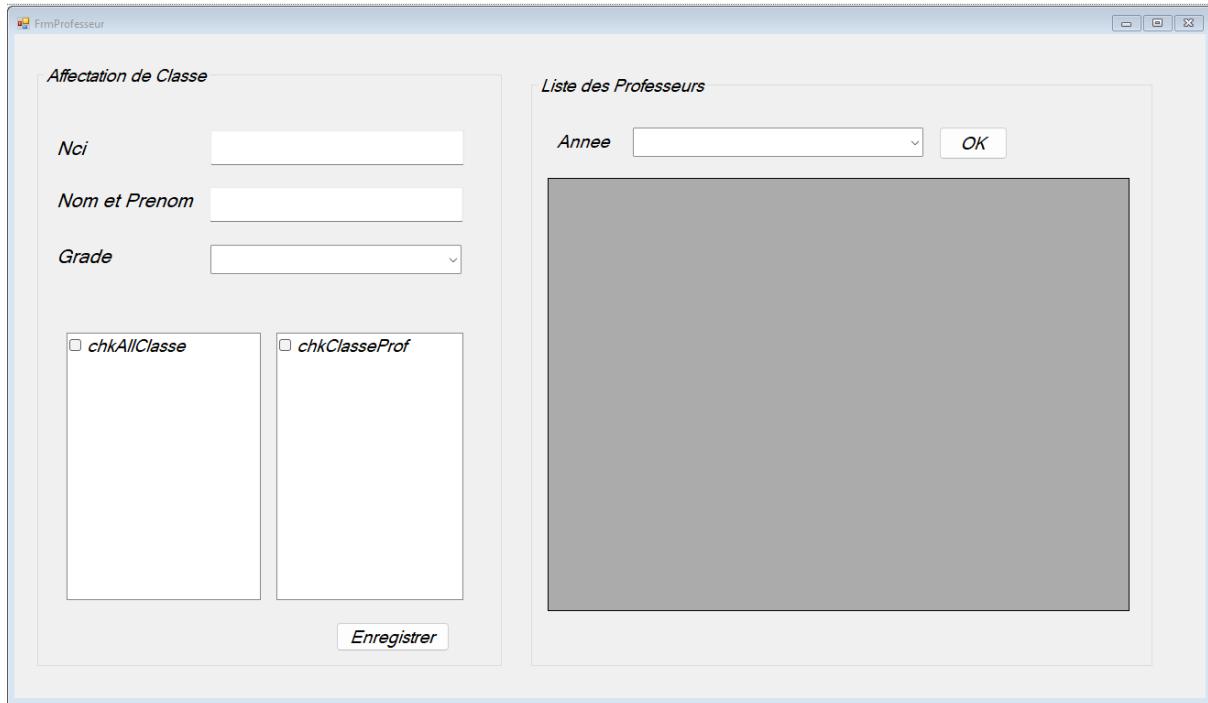
b) Charger le combo Annee dans la vue Prof

#views>IViewProf.cs

```
1 référence
AnneeScolaire Annees { get; set; }
```

#Presenter>PresenterProf.cs

O. Fonctionnalités de Professeur



1. Lister les Professeurs qui ont des classes dans l'année Encours

#Repository>ClasseRepository.cs

```

>List<Classe> FindByProfesseur(Professeur prof, AnneeScolaire annee);
    ↳ références
public List<Classe> FindByProfesseur(Professeur prof, AnneeScolaire annee)
{
    List<Classe> classes = new List<Classe>();
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        connection.Open();
        cmd.Connection = connection;
        cmd.CommandText = @"Select cl.id,cl.libelle,cl.filiere,cl.niveau from classes cl,
                           affectation aff where cl.id=aff.classe_id and aff.annee_id=@idAnn
                           and aff.professeur_id=@idProf ";
        cmd.Parameters.Add("@idAnn", SqlDbType.Int).Value = annee.Id;
        cmd.Parameters.Add("@idProf", SqlDbType.Int).Value = prof.Id;
        using (var reader = cmd.ExecuteReader())
        {
            if (reader.Read())
            {
                var classe = new Classe()
                {
                    Id = (int)reader[0],
                    Libelle = reader[1].ToString(),
                    Filiere = reader[2].ToString(),
                    Niveau = reader[3].ToString(),
                };
                classes.Add(classe);
            }
        }
    }
    return classes;
}

```

#Repository>ProfesseurRepository.cs

```
List<Professeur> FindProfesseurByAnnee(AnneeScolaire annee);

private IClasseRepository classeRepository;
1 référence
public ProfesseurRepository(string connectionString, IClasseRepository classeRepository)
{
    ConnectionString = connectionString;
    this.classeRepository = classeRepository;
}

U références
public List<Professeur> FindProfesseurByAnnee(AnneeScolaire annee)
{
    List<Professeur> professeursList = new List<Professeur>();
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        connection.Open();
        cmd.Connection = connection;
        cmd.CommandText = @"Select prof.id, nomcomplet,nci from users as prof where role like 'ROLE_PROFESSEUR' ";
        cmd.Parameters.Add("@id", SqlDbType.NVarChar).Value = annee.Id;
        using (var reader = cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                var professeur = new Professeur()
                {
                    Id = (int)reader[0],
                    NomComplet = reader[1].ToString(),
                    Nci = reader[2].ToString(),
                };
                if (professeur.Classes.Count() == 0)
                {
                    professeur.Classes = classeRepository.FindByProfesseur(new Professeur() { Id = professeur.Id }, annee);
                }
                professeursList.Add(professeur);
            }
        }
    }
    return professeursList;
}
```

#Service>Service.cs

```
1 référence
List<Professeur> listerProfesseur(AnneeScolaire annee);

public List<Professeur> listerProfesseur(AnneeScolaire annee)
{
    return professeurRepository.FindProfesseurByAnnee(annee);
}
```

#Views>IViewProfesseur.cs

```
AnneeScolaire Annees { get; set; }
2 références
String Grade { get; set; }
2 références
```

FormProfesseur.cs

```
dtgProfesseur.SelectionChanged += delegate { SeleledProfesseurEvent.Invoke(this, EventArgs.Empty); };

public void SetClasseBindingSource(BindingSource professeurList, BindingSource anneeList,
                                    BindingSource gradeList)
{
    cboAnnee.DataSource = anneeList;
    cboGrade.DataSource = gradeList;
    dtgProfesseur.DataSource = professeurList;
}

}
```

#Presenter>PresenterProfesseur.cs

```
private void initialize()
{
    List<ProfesseurDto> professeurs = service.listerProfesseur(anneeEncours);
    bindingSourceProf.DataSource = service.listerProfesseur(anneeEncours);
    bindingSourceAnnee.DataSource = service.listerAnnee();
    bindingSourceGrade.DataSource = service.listerGrade();
    this.view.AllClasses = service.listerClasse();

    this.view.SetClasseBindingSource(bindingSourceProf, bindingSourceAnnee, bindingSourceGrade);
}
```

2. Lister les Classes d'un Professeur

#Views>IViewProfesseur.cs

```
event EventHandler SeleledProfesseurEvent;
event EventHandler AffectationNewClasseEvent;
3 références
List<Classe> AllClasses { get; set; }
2 références
List<Classe> ClassesUnProf {set; }
```

FormProfesseur.cs

```
dtgProfesseur.SelectionChanged += delegate { SeleledProfesseurEvent.Invoke(this, EventArgs.Empty); };
btnDetails.Click += delegate { ShowClasseProfEvent.Invoke(dtgProfesseur, EventArgs.Empty); };
btnActive.Click += delegate {
    // ActiveEventForm.Invoke(this, EventArgs.Empty);
    txtNomPrenom.Enabled = true;
    txtNomPrenom.Clear();
    txtNci.Enabled = true;
    txtNci.Clear();
    cboGrade.SelectedIndex = 0;
};
```

```
public List<Classe> AllClasses {
    get
    {
        List<Classe> classes = new List<Classe>();

        foreach (var classe in chkAllClasse.CheckedItems)
        {
            classes.Add(classe as Classe);
        }
        return classes;
    }
    set {
        foreach (var classe in value)
        {
            chkAllClasse.Items.Add(classe);
        }
    }
}

< references
public List<Classe> ClassesUnProf {

    get
    {
        return chkClasseProf.Items;
    }
    set
    {
        chkClasseProf.Items.Clear();

        foreach (var classe in value)
        {
            chkClasseProf.Items.Add(classe);
        }
    }
}
```

#Presenter>PresenterProfesseur.cs

```
private void ShowClasseProfHandler(object sender, EventArgs e)
{
    ProfesseurDto prof = bindingSourceProf.Current as ProfesseurDto;
    // this.view.ClassesUnProf = prof.Classes;
    // DataGridView dataGridView = sender as DataGridView;
    // if (dataGridView.SelectedRows.Count == 0) return;
    FormClasseProfesseur frm = new FormClasseProfesseur(prof);
    frm.ShowDialog();
    frm.BringToFront();
    frm.StartPosition = FormStartPosition.CenterScreen;
}

1 référence
private void SeleledProfHandler(object sender, EventArgs e)
{
    ProfesseurDto prof = bindingSourceProf.Current as ProfesseurDto;
    view.ClassesUnProf = prof.Classes;
    view.Nci = prof.Nci;
    view.NomComplet = prof.NomComplet;
    view.Id = prof.Id.ToString();
}

}
```

3. Ajouter un Professeur et lui affecter des classes

#Repository>ProfesseurRepository.cs

```
void Add(Professeur prof,out int result);
2 références
void AddAffection(Professeur prof,Classe classe,AnneeScolaire annee);

public void Add(Professeur prof, out int result)
{
    result = 0;
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        try
        {
            connection.Open();
            cmd.Connection = connection;
            string sql = @"insert into users(nomCompletnom,role,ncl) values(@nom, 'ROLE_PROFESSEUR',@ncl)
                           + "SELECT CAST(scope_identity() AS int)";
            cmd.CommandText =sql;
            cmd.Parameters.Add("@nom", SqlDbType.NVarChar).Value = prof.NomComplet;
            cmd.Parameters.Add("@ncl", SqlDbType.NVarChar).Value = prof.Nci;
            if (sql.Trim().ToLower().StartsWith("insert"))
            {
                result =(int) cmd.ExecuteScalar();
            }
            else
            {
                cmd.ExecuteNonQuery();
            }
        }
        catch (SqlException ex)
        {
            throw new Exception(ex.Message);
        }
        finally
        {
            connection.Close();
        }
    }
}
```

```

< references
public void AddAffection(Professeur prof, Classe classe, AnneeScolaire annee)
{
    using (var connection = new SqlConnection(ConnectionString))
    using (var cmd = new SqlCommand())
    {
        try
        {
            connection.Open();
            cmd.Connection = connection;
            string sql = @"insert into affectation(classe_id,professeur_id,annee_id) values(@classe,@prof,@annee)";
            cmd.CommandText = sql;
            cmd.Parameters.Add("@classe", SqlDbType.Int).Value = classe.Id;
            cmd.Parameters.Add("@prof", SqlDbType.Int).Value = prof.Id;
            cmd.Parameters.Add("@annee", SqlDbType.Int).Value = annee.Id;
            cmd.ExecuteNonQuery();
        }
        catch (SqlException ex)
        {
            throw new Exception(ex.Message);
        }
        finally
        {
            connection.Close();
        }
    }
}

```

#Service>Service.cs

```

< references
void affacterClasse(Professeur prof, List<Classe> classes, AnneeScolaire anneeScolaire);
3 références
< references
public void affacterClasse(Professeur prof, List<Classe> classes, AnneeScolaire annee)
{
    if (prof.Id == 0)
    {
        professeurRepository.Add(prof, out int result);
        prof.Id = result;
    }

    foreach(var classe in classes)
    {
        professeurRepository.AddAffection(prof, classe, annee);
    }
}

```

#Views>IViewProfesseur.cs

```

3 références
String NomComplet { get; set; }
3 références
String Nci { get; set; }
3 références
String Id { get; set; }
event EventHandler NewProfEvent;

```

#Presenter>PresenterProfesseur.cs

```

this.view.NewProfEvent += NewProfEventHandler;

```

```

1 référence
private void NewProfEventHandler(object sender, EventArgs e)
{
    Professeur professeur=new Professeur()
    {
        Id =int.Parse(view.Id),
        Nci= view.Nci,
        NomComplet=view.NomComplet,
        Grade=view.Grade,
    };
    service.affecterClasse(professeur, view.AllClasses,anneeEncours);
    initialize();
}

```

#Views>FormProfesseur.cs

```

public string NomComplet { get => txtNomPrenom.Text.Trim();
    set {
        txtNomPrenom.Enabled=false;
        txtNomPrenom.Text = value;
    }
}
3 références
public string Nci { get => txtNci.Text.Trim(); set => txtNci.Text = value; }
3 références
public string Id { get => txtID.Text.Trim(); set => txtID.Text=value; }

```

4. Ajouter un Professeur et lui affecter des classes

```

private void NewProfEventHandler(object sender, EventArgs e)
{

    ProfesseurDto professeur=new ProfesseurDto()
    {
        Id =int.Parse(view.Id),
        Nci= view.Nci,
        NomComplet=view.NomComplet,
        Grade=view.Grade,
    };

    service.affecterClasse(professeur.toProfesseur(), view.AllClasses,anneeEncours);
    initialize();
}

```

#Design Pattern Builder ProfDto

5. Validation de l'entité Professeur

```
public class ProfesseurValidate : ValidationAttribute
{
    0 références
    public ProfesseurValidate() : base("{0} existe deja")
    {

    }

    0 références
    protected override ValidationResult IsValid(object value, ValidationContext validationContext)
    {
        bool valid = Fabrique.GetInstance().UserExistByNci(value.ToString());
        if (valid)
        {
            return new ValidationResult(base.FormatErrorMessage(validationContext.MemberName)
                , new string[] { validationContext.MemberName });
        }

        return null;
    }
}
```

ProfesseurDto.cs

```
[System.ComponentModel.DisplayName("Nom et Prenom")]
[Required(ErrorMessage = "Le Login est Obligatoire")]
3 références
public string NomComplet { get => nomComplet; }

[System.ComponentModel.DisplayName("Profil")]
0 références
public Role RoleUser { get => role; set => role = value; }

[System.ComponentModel.DisplayName("NCI")]
[Required(ErrorMessage = "Le Login est Obligatoire")]
[ProfesseurValidate]
2 références
public string Nci { get => nci; }

4 références
public List<Classe> Classes { get => classes; }

[System.ComponentModel.DisplayName("Grade")]
[Required(ErrorMessage = "Le Login est Obligatoire")]
0 références
public string Grade { get => grade; }
```

41 - LLL-----

Projet

La clinique 221 fait appel à vous pour la réalisation d'une application WinForm de gestion des rendez-vous de consultation ou d'analyse . La clinique offre des services tels des consultations avec un médecin généraliste ou spécialiste(dentiste , ophtalmologie,..) et des prestations(Analyse,Radio,etc..).

Une consultation est caractérisée par la date, le médecin , les constantes qui ont été prises(Température, tension,etc..) et éventuellement les prestations ou l'ordonnance qui ont été prescrites au patient.Une ordonnance est constituée de plusieurs médicaments(code, nom) et de la posologie.

A la fin d'une consultation, le médecin peut planifier un autre rendez-vous pour le patient(code, nom,prénom et antécédents médicaux{Diabète,Hypertension,Hépatite}).

Un Médecin à la possibilité de lister ses rendez-vous de consultation de la journée , filtrer par date,et annuler une consultation,de rechercher le dossier médical d'un patient(l'ensemble des consultations et des prestations effectuées par un patient dans la clinique), voir les détails d'une consultation.Il peut annuler un Rendez vous de consultation.

La secrétaire crée les demandes de rendez-vous pour une consultation ou une prestation .

Il peut lister les consultations ou les prestations de la journée , filtrer par date ou par patient .

Il peut aussi annuler un rendez-vous de consultation ou une prestation à la demande du patient.Cette demande doit être faite 48H avant la date du rendez-vous .

Un rendez-vous de consultation est validé s'il y a une disponibilité du médecin demandé ou de la prestation .

Le Responsable des prestations peut lister les Rendez-vous de prestations de la journée, voir les détails d'une prestation,filtrer par date,et annuler .

Lorsqu'une prestation est faite , elle est validée et les résultats sont importés.

On voudrait avoir les statiques suivantes:

- Les Médecins disponible par jour ,
- Les Prestation du Jour
- Les Consultation du jour ,avec la possibilité de filtrer
- Les Consultation annulées du Jour

NB:Toutes les fonctionnalités sont accessibles après connexion

TAF :

I. Modélisation

- A. Faire le Diagramme de Use Case
- B. Faire le Diagramme de Classe
- C. Faire le Diagramme de Séquence de Chaque Use Case
- D. Faire le MLD
- E. Créer la Base de données ainsi que les tables

II. Réalisation du Projet

- A. Créer le projet
- B. Créer la Structure du Projet avec l'architecture MVP