# Mission 3 – Mise en place environnement

[I Architecture générale 1](#__RefHeading___Toc5055_3472932569)

[II Préparation des machines 2](#__RefHeading___Toc4743_3472932569)

[1 Description des machines 2](#__RefHeading___Toc4745_3472932569)

[2 Mise à jour des paquets 2](#__RefHeading___Toc4747_3472932569)

[III Préparation de Mutillidae 2](#__RefHeading___Toc4749_3472932569)

[1 Présentation de Mutillidae 2](#__RefHeading___Toc4751_3472932569)

[2 Installation de Mutillidae 3](#__RefHeading___Toc4753_3472932569)

[2.1 Installation 3](#__RefHeading___Toc4755_3472932569)

[2.2 Initialisation de la configuration 5](#__RefHeading___Toc4757_3472932569)

[IV Préparation de BurpSuite 6](#__RefHeading___Toc4759_3472932569)

[1 Présentation de BurpSuite 6](#__RefHeading___Toc4761_3472932569)

[2 Installation de BurpSuite 6](#__RefHeading___Toc4763_3472932569)

[2.1 Installation 6](#__RefHeading___Toc4765_3472932569)

[2.2 Configuration du navigateur 8](#__RefHeading___Toc4767_3472932569)

[2.3 Première capture de paquet 9](#__RefHeading___Toc1960_1299089629)

[V Pistes d’exploitation pédagogique 11](#__RefHeading___Toc4771_3472932569)

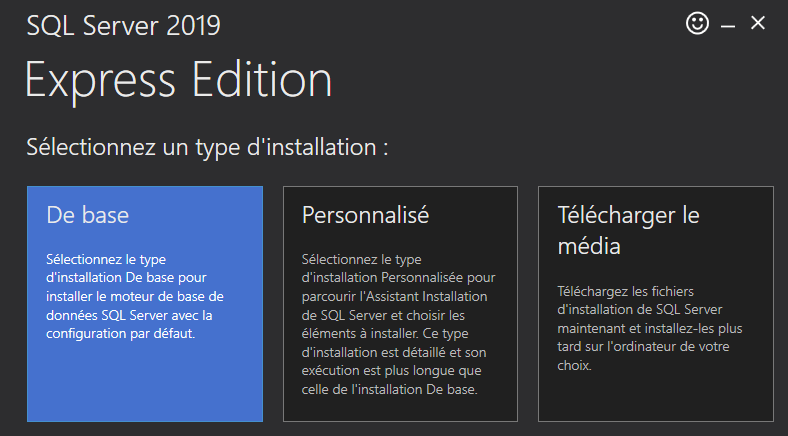
#### Préparation de la base de données

##### Installation de sql serveur en local

Nous allons installer Sql serveur en local. Pouvez récupérer .exe d’installation de depuis le serveur sur la zone énoncé.



Suivez les instructions de l’assistant d’installation.

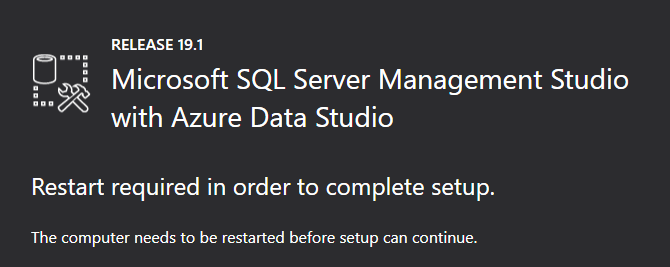


Menez la procédure d’installation jusqu’au bout.

Nous allons ensuite installer le client permettant d’utiliser la base de données Sql Serveur, à savoir *Sql Server Management Studio*.



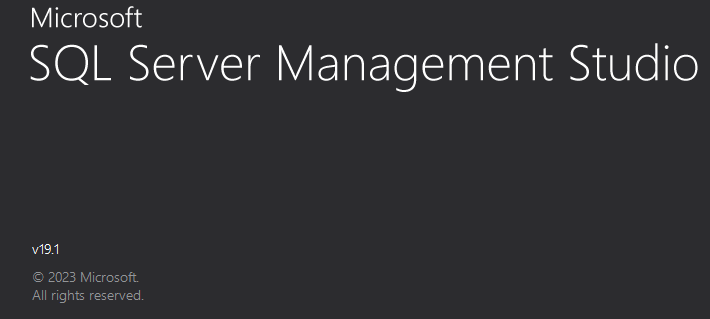
Suivez les instructions de l’assistant d’installation.



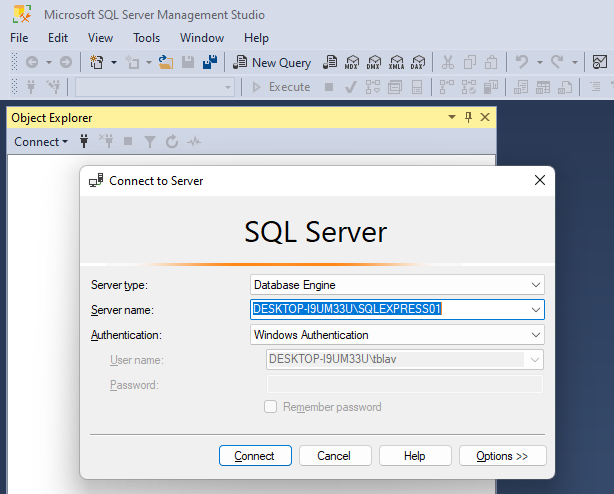
A la fin de l’installation, nous allons le logiciel pour installer la base de données.



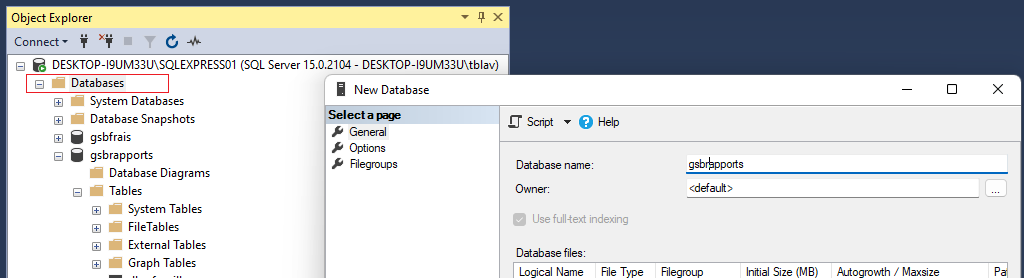
Nous obtenons la mire suivante :



Au lancement de l’application, nous allons nous connecter en mode *authentification windows*.



Nous allons créer une nouvelle base de données gsbrapports. Faire un clic droit sur databases, puis sélectionné *new database*.

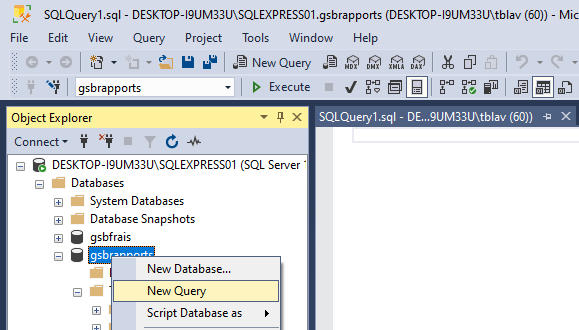


Nous allons maintenant créer alimenter notre nouvelle base de données.

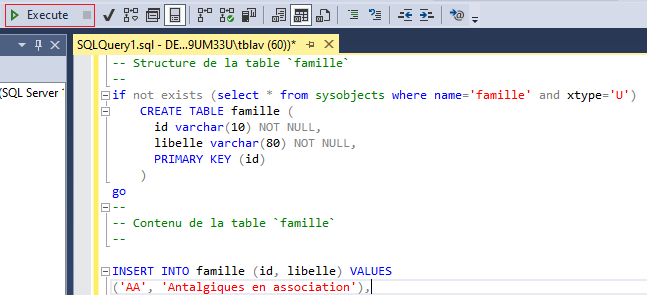
Vous trouverez un fichier nommé *gsb2023\_sqlserver.sql,* qui contient le script d’alimentation de la base de données.



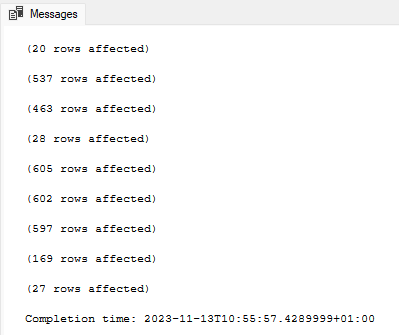
Nous allons charger le script, clic droit sur la base de données, puis cliquer sur *new query*.



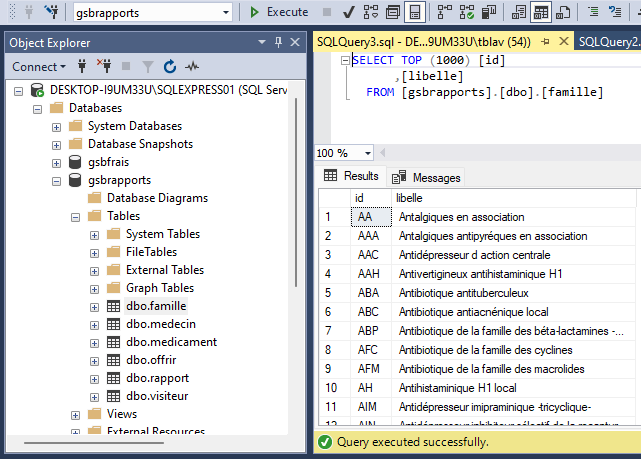
Chargez le fichier *gsb2023\_sqlserver.sql* dans l’onglet.



Puis cliquez sur *Execute*



Vérifier que votre base est bien alimentée.



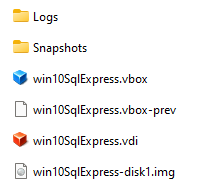
Exemple : exécution d’une requête

##### 2 Utilisation d’’une machine virtuelle avec sql serveur

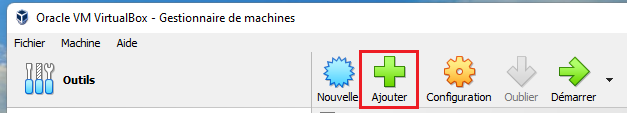
Une machine virtuelle est nécessaire servant à héberger le serveur une base de données Sql Serveur.



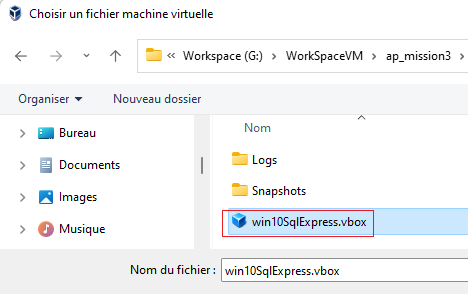
Vous pouvez récupérer le dossier *win10SqlExpress* d’installation de depuis le serveur sur la zone énoncé.



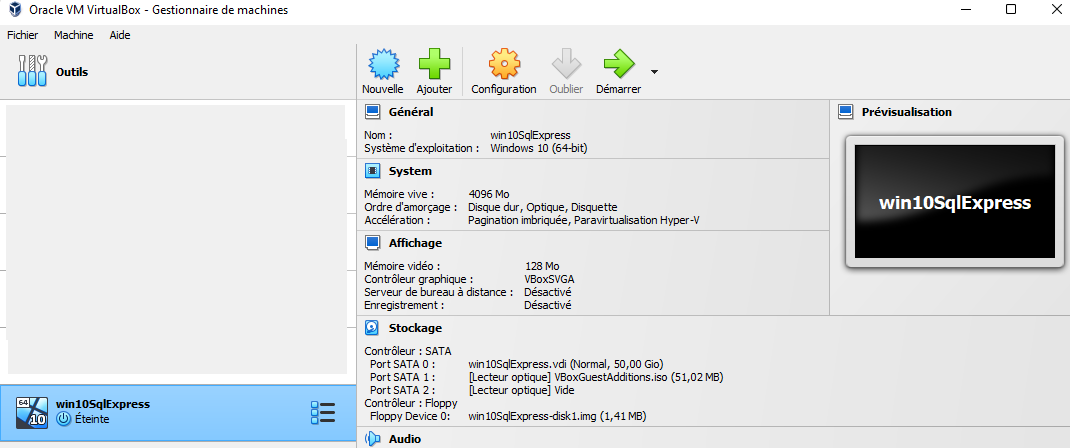
Il est fortement recommandé de réaliser ces installations à l'aide d'un outil de virtualisation (VMWare, VirtualBox…).



Cliquez sur *Ajouter*

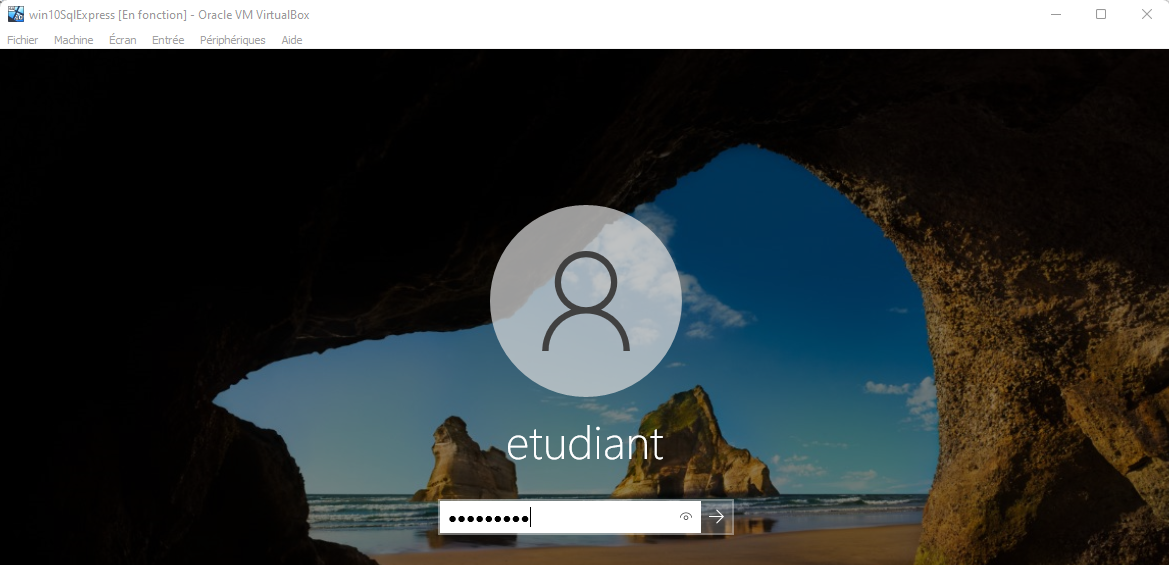


Cela va automatiquement créer une nouvelle machine virtuelle *win10SqlExpress.*



Veuillez-vous assurez des paramètres de votre machine virtuelle comme vu dans le TP9 en D5 sécurité.

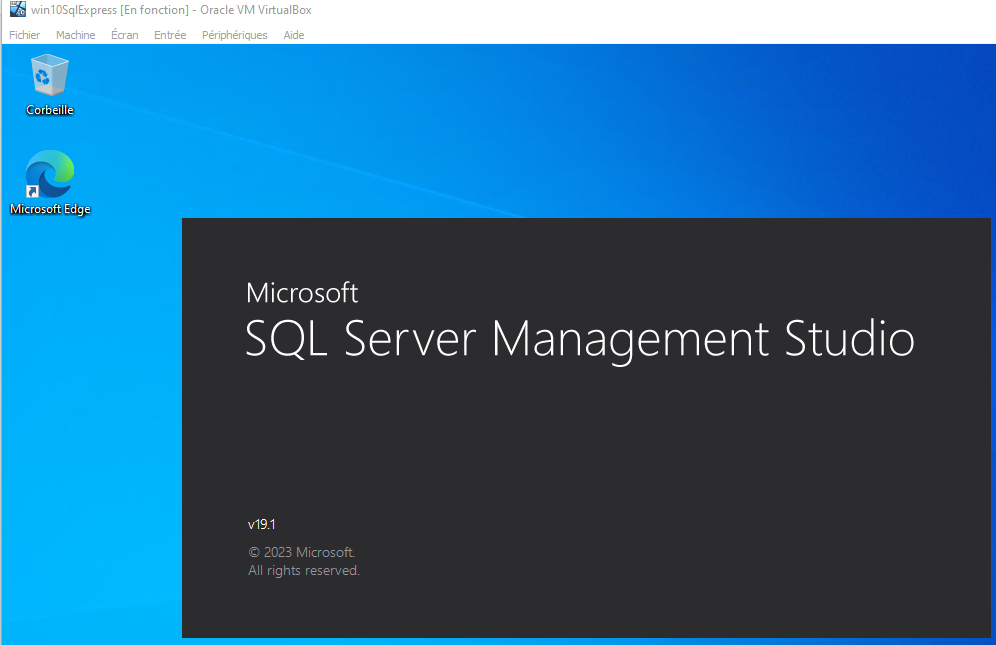
Assurez vous de mettre en place un partage réseau avec la machine virtuelle. Puis lancer votre VM.



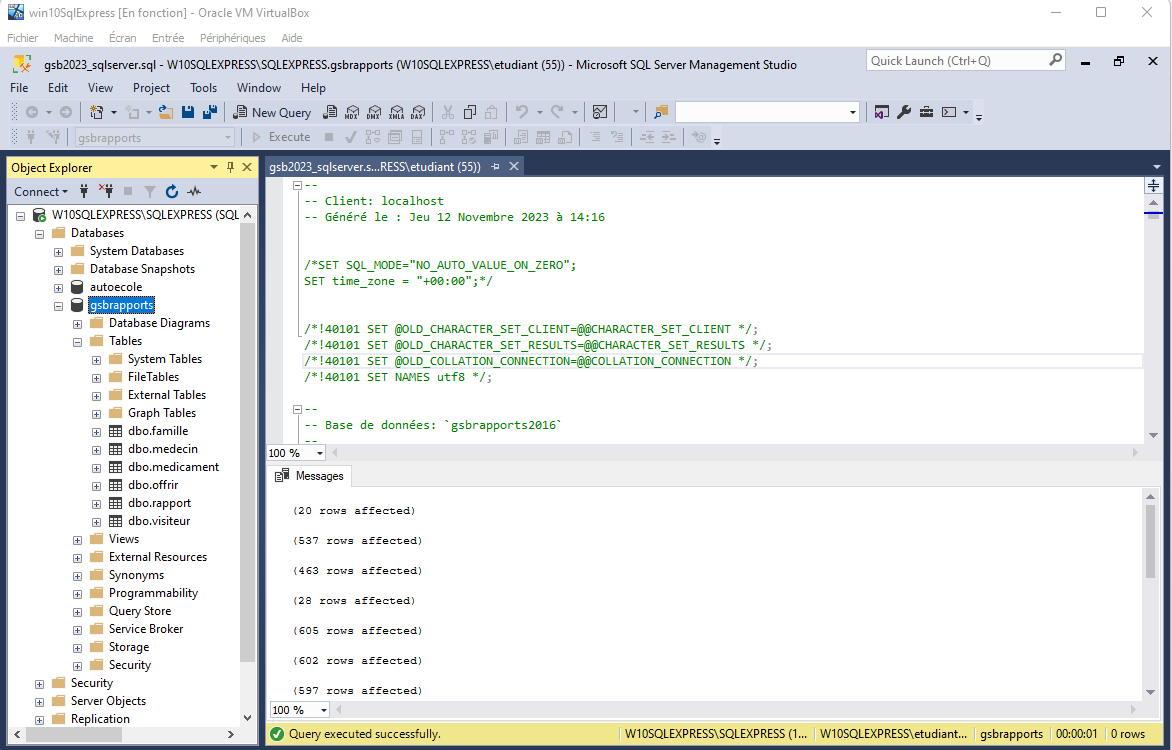
Login : etudiant

Mot de passe : Etudiant

Veuillez lancer l’application *Sql Server Management Studio*.



Installer la base de données *gsprapports* comme dans la procédure d’installation en locale.



Veuillez noter le nom du serveur.

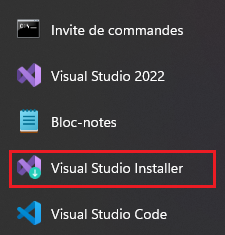
*Nom du serveur : W10SQLEXPRESS\SQLEXPRESS*

Votre installation est terminée, vous pouvez lancer Visual Studio.

#### Préparation de l’environnement sous Visual Studio

##### Prérequis configuration sous Visual Studio

Lancer Visual studio installer pour vérifier votre configuration.

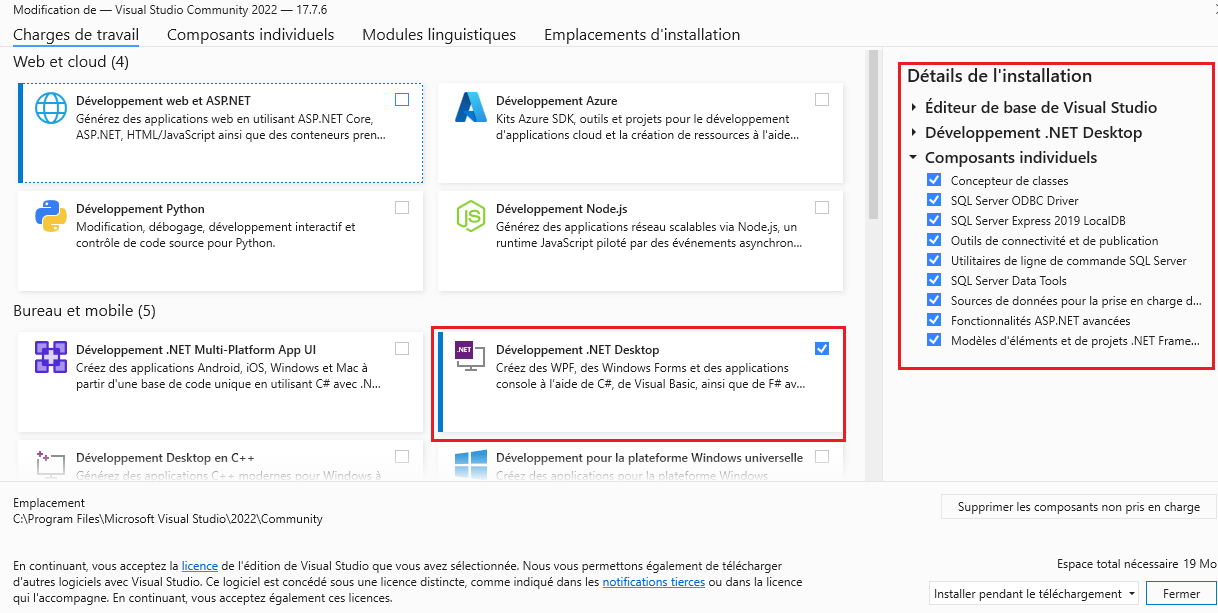


Vous devez vérifier l’installation des composants suivants :

* Modèles d’éléments et projets .Net Framework
* Developpement .Net Desktop
* Sources de données pour le prise en charge SQL Server



Cliquez sur modifier pour vérifier ou mettre à jour votre environnement.

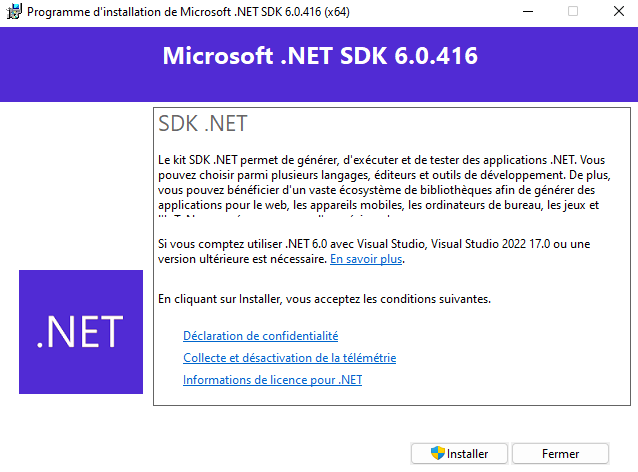


##### Installation de Framework 6.0

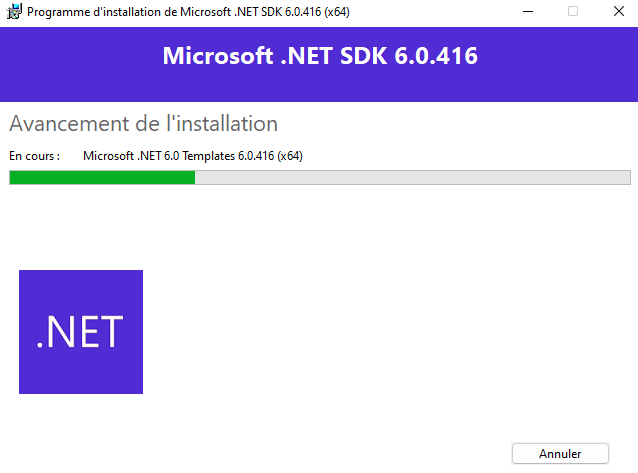
Vous devez récupérer le fichier installation sur le serveur.



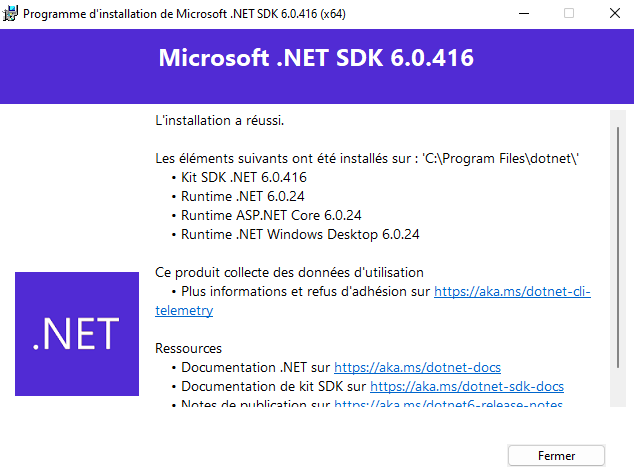
Ensuite lancer l’installation



Suivez les instructions de l’assistant d’installation.



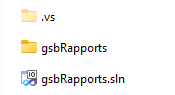
Fin de l’installation



##### 2 Configuration du projet sous Visual Studio

Nous allons configurer Visual Studio pour configuration du projet.

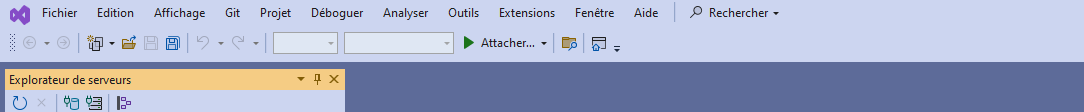
Dans un premier récupérer le dossier *gsbRapports* contenant la structure du projet.



Lancer Visual Studio.



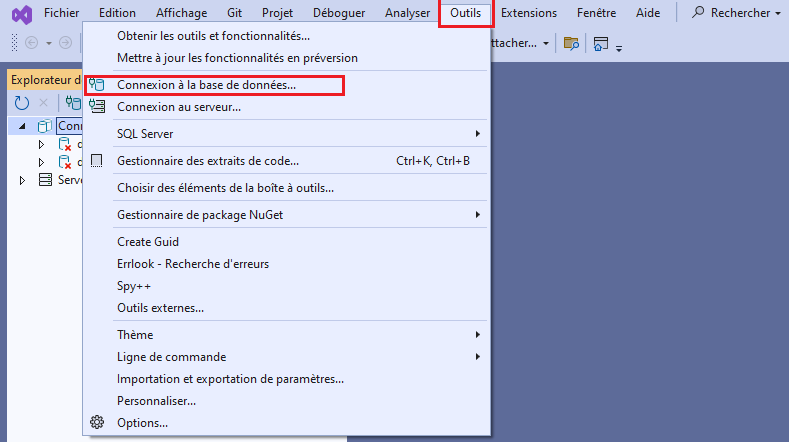
Nous allons ouvrir un projet / solution



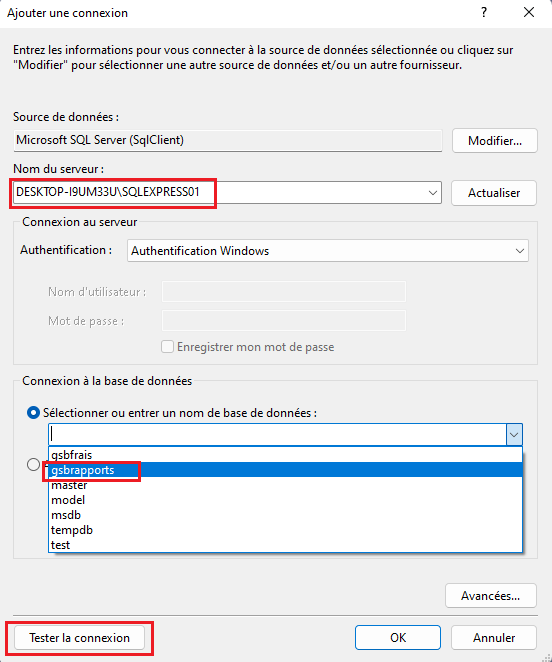
##### Connecter la base de données dans Visual Studio

###### 2.1 Mise en place base de données

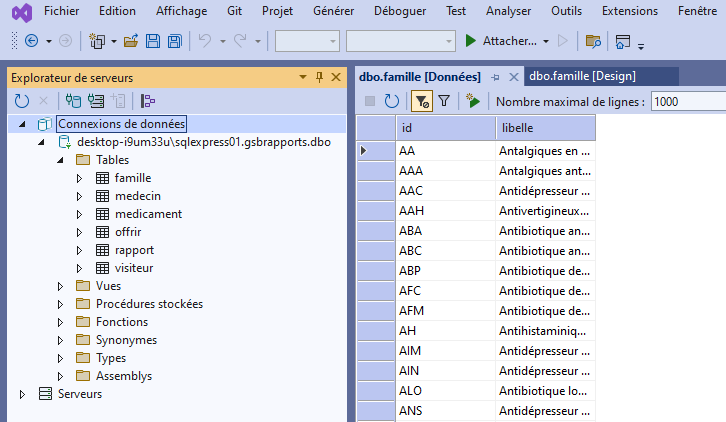
Nous allons connecter la base dans Visual studio.



Vous devez renseigner le *nom du serveur* de base de données, le *nom de la base* sur laquelle vous souhaitez vous connecter et enfin faire un *Tester la connexion*.



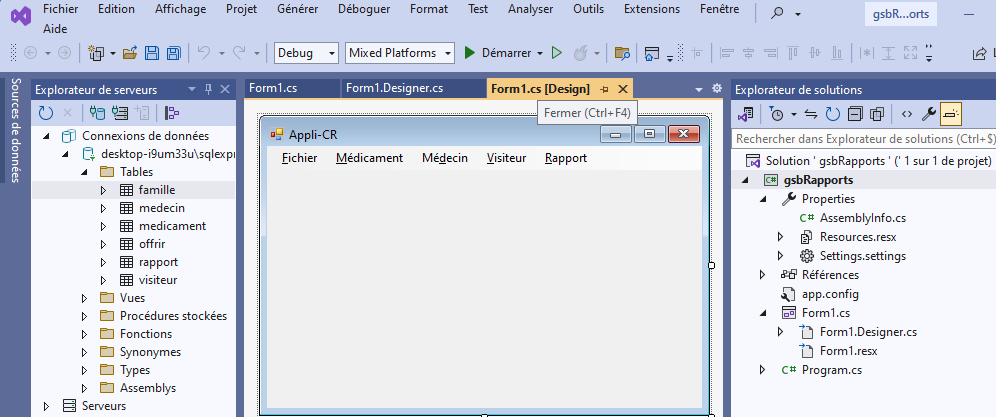
Vérifier que votre base est bien alimentée.



Nous ainsi passer à la configuration du projet.

###### 2.2 Configuration du projet

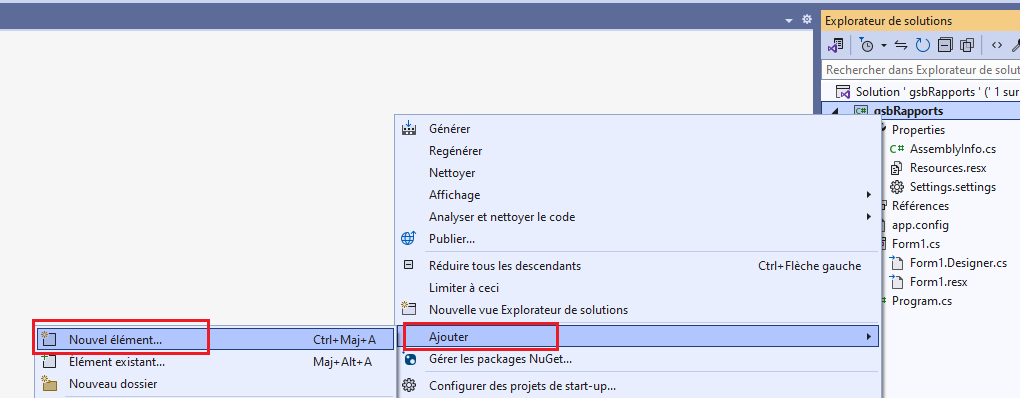
Nous allons charger le projet *gsbrapports* dans Visual studio.



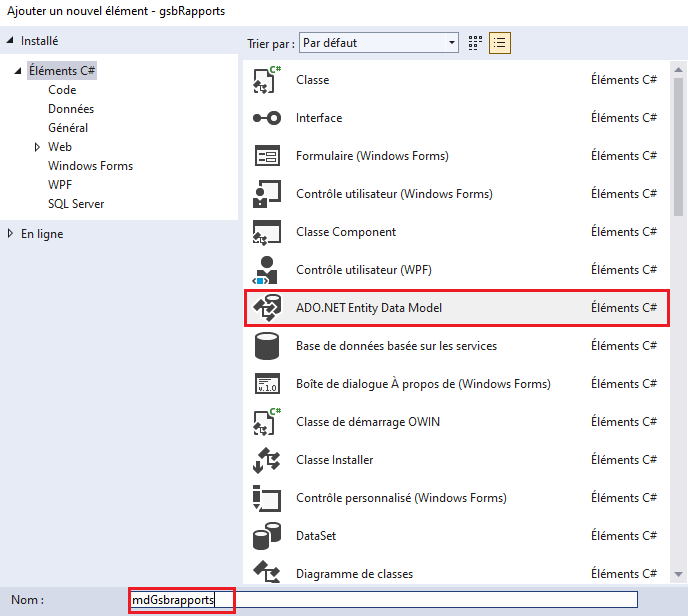
Vous devez avoir la représentation ci-dessus.

###### 2.3 Création d’un projet

Nous allons charger le projet *ADO*.

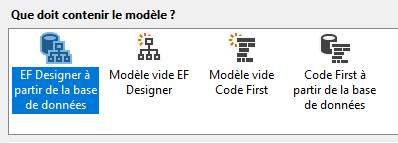


Ajouter au projet un nouvel élément ADO



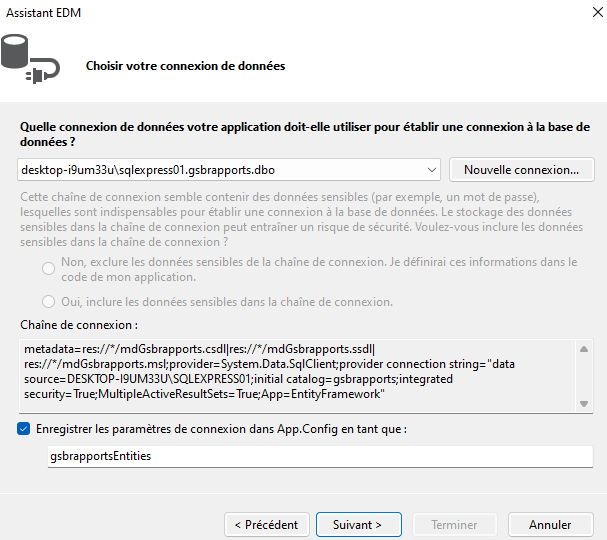
Cet élément utilise une bibliothèque de classe (un Framework Entity Framework –EF-) dédiée à l’accès aux données.

Un assistant va générer ensuite des classes permettant cet accès aux données.



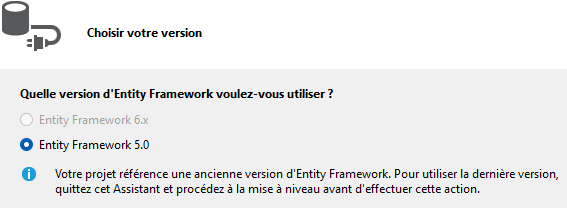
Configurer la connexion :

1. Sélectionner la base de données *gsbrapports* dans la liste.
2. Tester.
3. Faire suivant et accepter le nom de la connexion proposée.

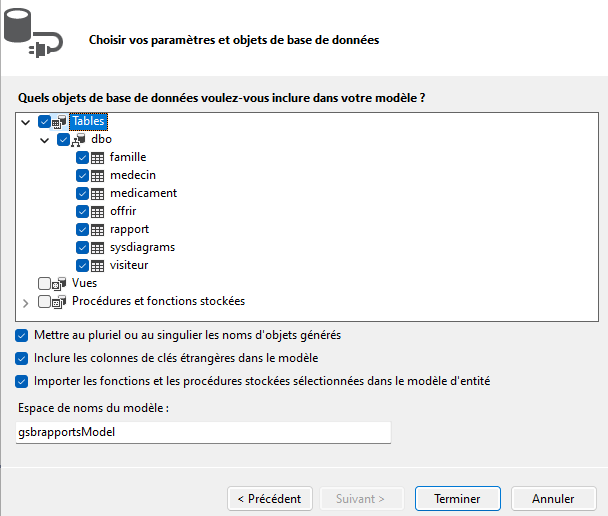


L’assistant va ensuite vous demander de sélectionner les tables visées :

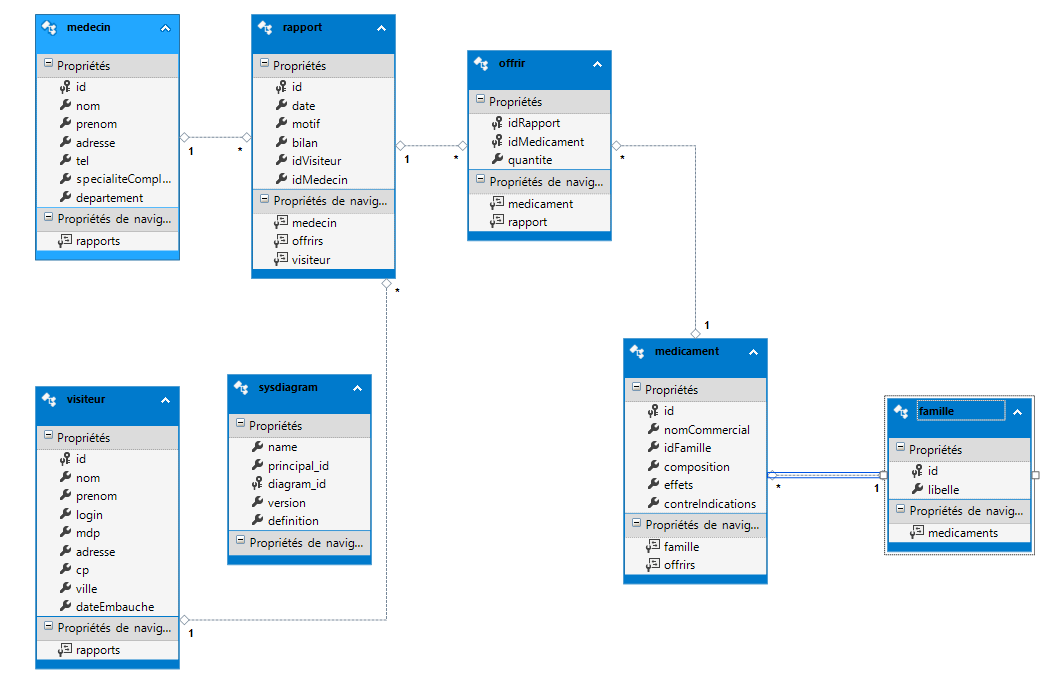
1. Cocher bien les deux cases.
2. Modifier le nom du namespace proposé.
3. Faire terminer.



Suivez les instructions de l’assistant d’installation.



Entity Framework génère un modèle objet qui sera chargé en mémoire :



Chaque table devient ainsi une classe. Les propriétés de navigation permettront de naviguer de classe en classe, si on le souhaite. Notez le singulier ou le pluriel sur ces propriétés.

Ainsi un rapport concerne un seul ***medecin*** et un seul ***visiteur*** ; par contre un ***medecin*** a plusieurs rapports, et le ***visiteur*** aussi.

Remarquons également que la classe ***rapport*** contient des champs correspondant aux clés étrangères : ***idVisiteur*** et ***idMedecin*** ; nous pouvions ne pas générer cette présence des clés étrangère en décochant plus haut la case Inclure les colonnes clés étrangères…

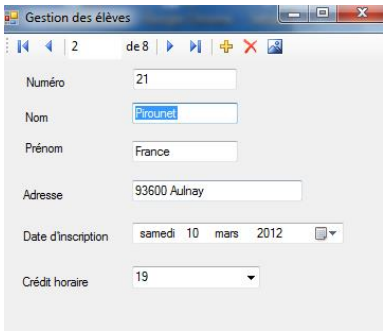
Regardons le fichier ***XML App.Config***. C’est dans ce fichier que se trouvent les paramètres de connexion.

**Annexe 1 : Exemple de création formulaire winform**

### Le formulaire Elève/Gérer

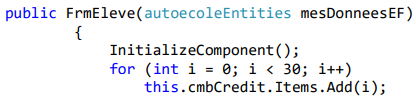
* Ajouter au projet un nouveau formulaire, FrmEleve.

Il doit ressembler à ceci :

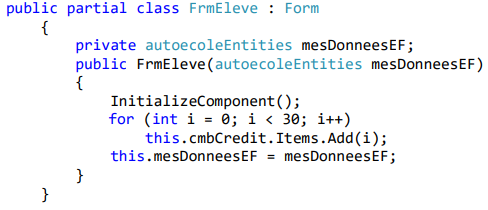


Le champ de type date est un DateTimePicker, le crédit horaire un ComboBox, ne tenez pas compte de la barre de navigation pour l’instant.

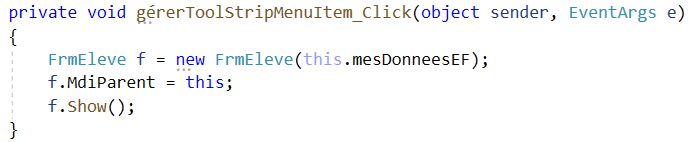
* Réaliser en conception ce formulaire, attention au nommage des composants :  
  txtNum, txtNom, …, dtInscription, cmbCredit
* Le comboBox proposant le crédit horaire doit être chargé par une boucle :



* Nous allons ensuite lier l’ouverture de ce formulaire au formulaire de menu :



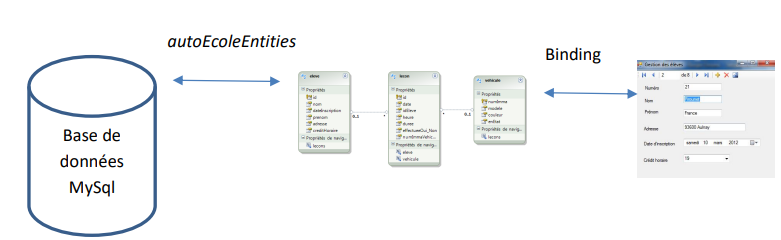
* Dans le formulaire FrmMenu :



#### Notion de binding

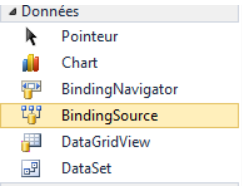
Le binding est un mécanisme qui permet de lier un composant graphique de présentation à une source de données. Cette liaison est bi-directionnelle, si l’on modifie la source, l’information sera transmise au composant et si on modifie l’information présente dans le composant, ceci sera répercuté dans la source de données. Une source de donnée peut être de nature très variée, ce peut être une table d’une base, une classe en mémoire, une liste typée. Dans notre cas, il s’agit d’une classe en mémoire.

On peut représenter les différentes responsabilités ainsi :



Le binding est mis en œuvre grâce à un composant qu’il faut installer dans le formulaire.

Il se trouve dans la partie Données de la boite à outils :



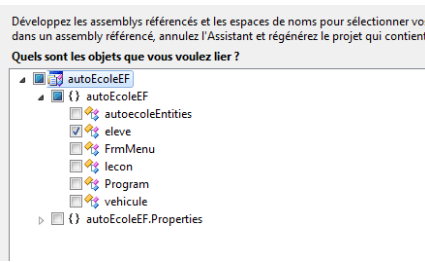
Déposez-le sur le formulaire ; il n’apparait pas dessus, mais en dessous.

Ce composant n’est pas un composant graphique.

* Il faut maintenant le renommer (bdgSourceEleve) et lui indiquer sa source de données qui est la classe eleve (et pas, bien sûr la table eleve).
* Dans les propriétés du composant, cliquer sur DataSource, ajouter une nouvelle source de données.
* Indiquer que la source est un objet.

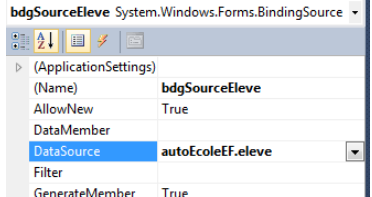


Sélectionner la classe eleve.



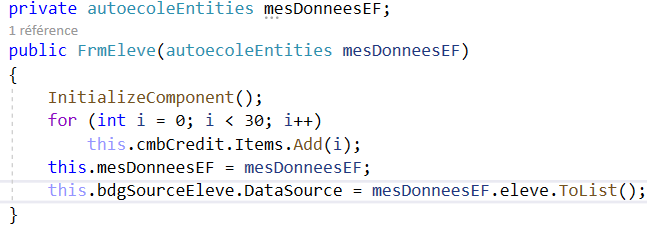
* Si la source n’apparait pas, compilez le projet.

Le composant de binding connait maintenant sa source de données :



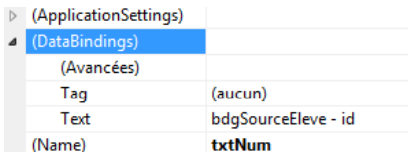
Remarque : la liaison se fait sur la classe (en fait Entity selon les termes de MS),  
**il n’y a pas encore de chargement des données venant de la classe de contexte de donnée**.

Ce chargement des données se fera par le code :



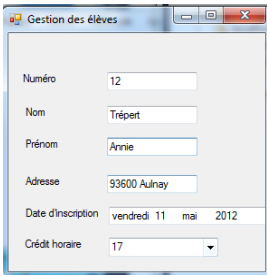
Nous venons donc de lier le composant de binding à la source de données ; il faut maintenant lier le composant graphique et le composant de binding.

C’est ce que nous ferons par la propriété DataBinding/text du premier textBox (celui qui contient l’id de l’élève) que nous lierons à l’id de bdgSourceEleve pour son champ id :



Faites de même pour toutes les propriétés Text des composants du formulaire.

Si nous lançons l’application, on constate l’accès aux données

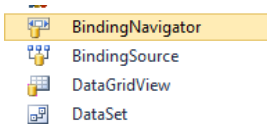


**Naviguer dans la table**

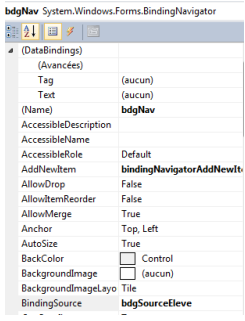
Nous allons utiliser un composant dédié à la navigation, le BindingNavigator.

Déposer le sur le formulaire.

Le formulaire fait apparaître une barre de navigation.



* Modifiez son nom en bdgNav
* Paramétrons ce composant dans sa propriété BindingSource

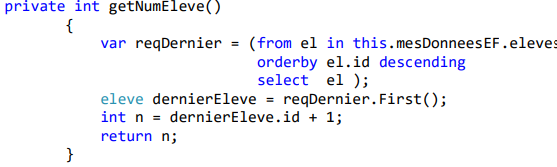


Si nous testons l’application, nous remarquons la possibilité de modifier, supprimer un élève.

Mais si nous relançons l’application, la base de données n’a pas été modifiée ; pour cela, il faut demander explicitement à la classe qui communique avec la base de la faire, la classe autoEcoleEntities. On le fera un peu plus tard.

Intéressons-nous au bouton d’ajout du navigateur. Se pose le problème de l’identifiant, il n’est pas auto-incrémenté dans la base, il doit prendre une valeur ici (d’autres solutions pouvaient être envisagées –procédure stockée-). Nous allons générer un numéro qui incrémente le numéro maximum existant.

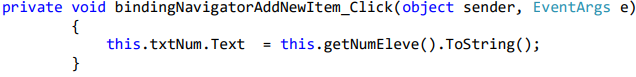
* Écrire la méthode getNumEleve dans la classe FrmEleve.



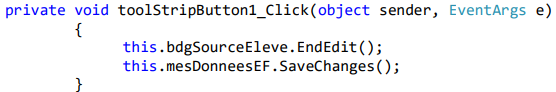
On voit ici ce qu’on appelle une requête linq, langage d’interrogation des données pour les classes de mapping. L’avantage est de bénéficier de l’IntelliSense lorsqu’on requête, le prix à payer c’est le nouveau langage de requêtage !!

Mettre le champ txtNum à readOnly = true (l’utilisateur ne peut ainsi pas modifier l’identifiant) on va faire un appel à cette méthode lorsque l’on désire ajouter un nouvel élève.

Ouvrez le code l’événement clic du bouton ajouter du navigateur :



Ajoutez un bouton à la barre de navigation, dans sa page de propriété indiquons « enregistrer » à sa propriété Text et dans le code de l’événement clic associé :

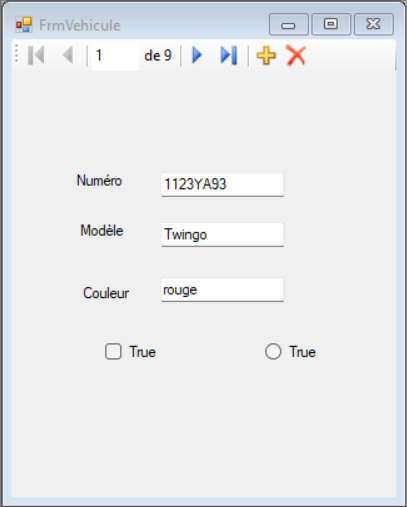


L’instruction endEdit() permet de valider dans le modèle de classe la création. Cet appel doit se faire explicitement parfois lorsque nous voulons faire prendre en compte immédiatement une modification. Cet appel est le plus souvent fait automatiquement.  
L’appel à SaveChanges() se justifie ici, car la méthode getNumEleve est une requête linq qui va chercher en base un nouveau numéro ; si nous n’enregistrons pas en base ce numéro, la requête retournera toujours la même valeur ;

### Le formulaire Véhicule/Gérer

En vous inspirant du formulaire précédent, créer ce formulaire et le code associé. Contrainte : les champs doivent être en readOnly, sauf si on ajoute un véhicule.

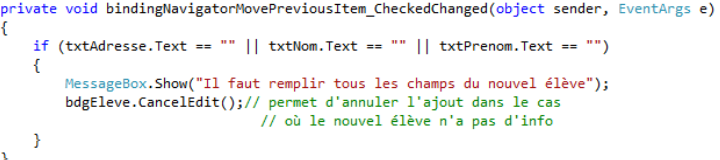
Voici la maquette attendue :



#### Un petit bug

Lorsque l’on ajoute un nouveau véhicule (il en était de même pour l’ajout d’un nouvel élève) et que l’on ne saisit pas les informations, l’application actuellement ne signale pas d’erreur. Nous allons modifier ce comportement. L’idée, c’est que, si l’utilisateur « sort » de l’ajout (en cliquant sur l’un des 3 boutons disponibles, movePrevious, moveFirst, addNew) sans avoir rempli les champs nécessaires, l’application le signale et annule la demande en cours.

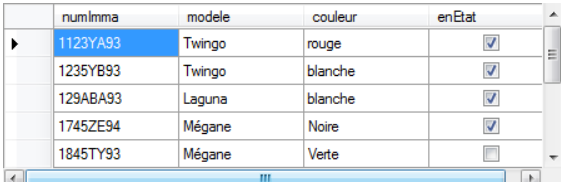
1. Il faut dans un premier temps positionner la propriété CheckOnClicK à true pour les trois boutons énumérés plus haut, ce qui n’est pas fait par défaut. Ainsi, par défaut, si on clic sur un bouton, il n’est pas checké ! On peut regretter ce comportement par défaut.
2. Il faut « intercepter » l’événement lorsque le bouton est « dé-chécké », événement qui a lieu avant l’événement click. C’est l’événement checkedChanged.



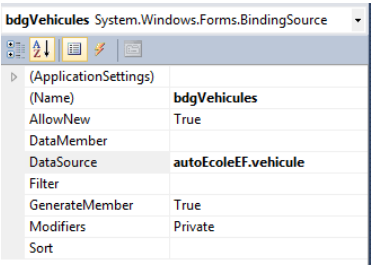
Il faut procéder de même pour les deux autres boutons concernés, moveFirst, addNew. La méthode CancelEdit annule la dernière modification.

### Le formulaire Véhicule/Liste

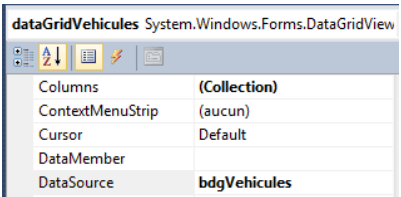
Ce formulaire permet de visualiser tous les véhicules et éventuellement modifier le champ enEtat. Voici la maquette attendue :



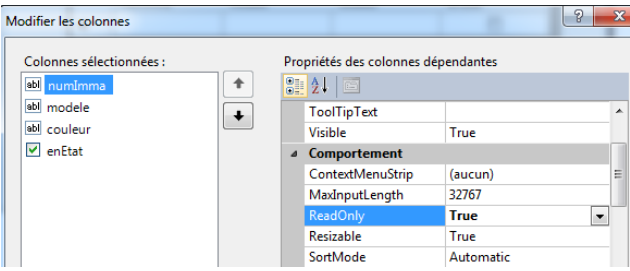
1. Créez un nouveau formulaire FrmListeVehicule,
2. liez-le à l’ouverture de l’option Véhicule/Liste du menu principal,
3. modifiez son constructeur,
4. passez le champ privé dans l’appel de la construction,
5. ajouter un composant de binding : bdgVehicules,
6. créez une nouvelle source de données pour valoriser son champ DataSource.



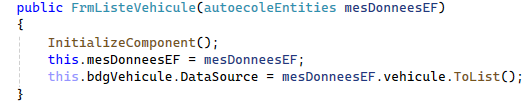
1. Déposez dans le formulaire un DataGridView,
2. renommez-le,
3. valorisons sa propriété DataSource :



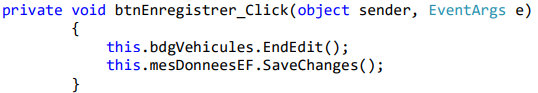
1. Sur un clic droit, sur le DataGridView, modifiez les colonnes pour rendre les 3 premières en readOnly :



1. Ajoutez le chargement des données dans la construction :

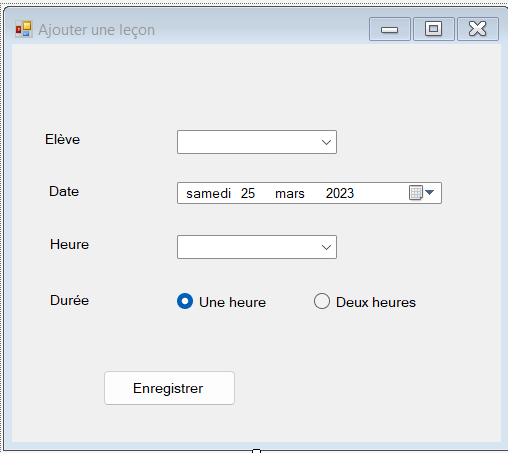


1. Enfin, ajoutez un bouton dans le formulaire pour enregistrer les modifications :



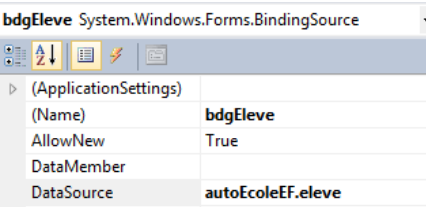
### Le formulaire d’ajout d’une leçon

Voici la maquette attendue :

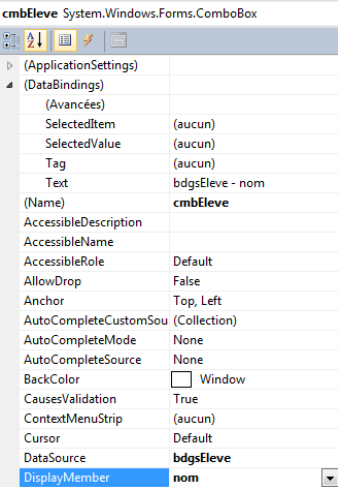


En effet, le véhicule sera ajouté plus tard ainsi que l’enregistrement du champ effectuee\_oui\_non.

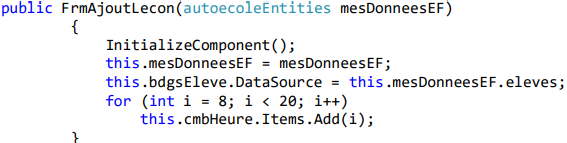
1. Créez ce formulaire et liez-le au menu, comme déjà fait plus haut,
2. liez le comBox des élèves à la source de données des élèves,
3. déposez un composant de binding bdgEleve lié à la source de données déjà créée



1. Liez le comboBox cmbEleve à cette source :

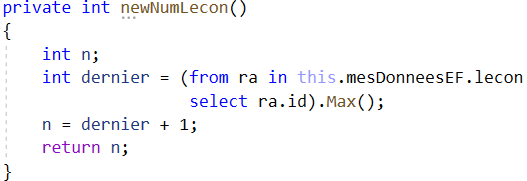


1. Remplissez la source de données et chargeons cmbHeure avec des données correspondant à des horaires

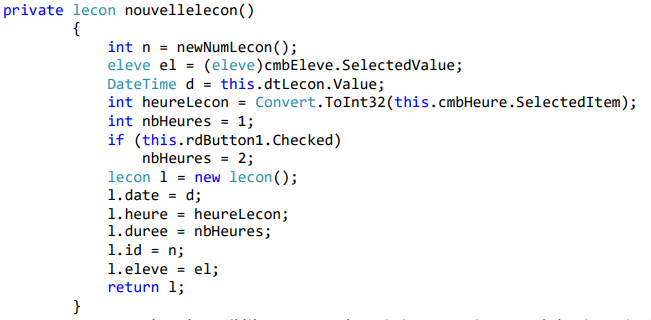


Pour enregistrer une nouvelle leçon, il nous faut faire comme pour un nouvel élève, générer un id incrémenté.

1. Mettez en œuvre une requête plus performante :



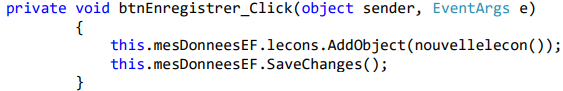
1. Générez un objet de type lecon



Remarques : nous récupérons l’élève concerné par la leçon par la propriété SelectedValue.

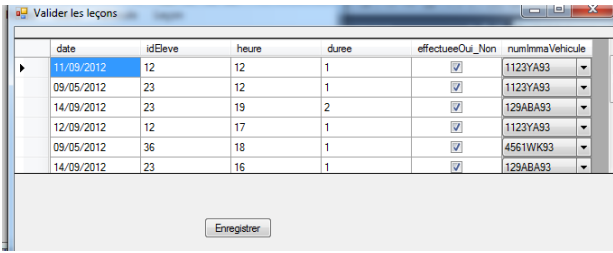
Le constructeur généré pour la classe lecon est le constructeur par défaut (sans argument) ; c’est pourquoi nous employons des seter pour valoriser les champs. Nous aurions pu aussi surcharger ce constructeur.

1. Enregistrez la nouvelle leçon par l’appel du clic sur le bouton enregistrer



### Le formulaire Leçon/Valider

Il s’agit ici de fixer le numéro d’immatriculation du véhicule et indiquer que la leçon a eu lieu ou pas. Voici la maquette attendue :

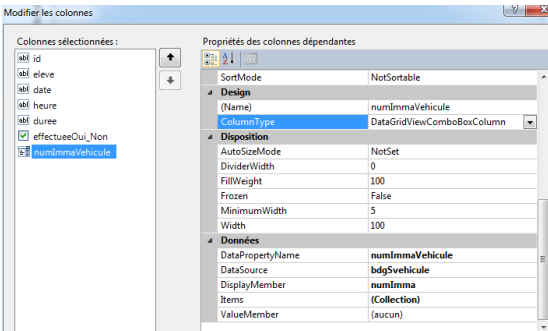


Procédez comme pour les autres formulaires :

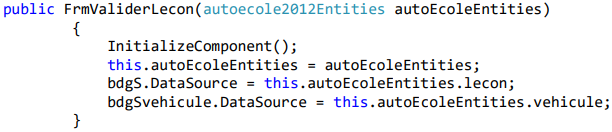
1. Ajoutez un formulaire FrmValiderLecon,
2. liaison au menu,
3. déclaration du champ de contexte de données,
4. modification du constructeur.
5. Ajoutez un composant de binding bdgSlecon
6. liez ce composant à la classe lecon,
7. Ajoutez un DataGridView
8. liez-le par sa propriété DataSource au composant de binding bdgSlecon

**Modification des colonnes du DataGridView**

1. ajoutons un second composant de binding bdgSvehicule,
2. liez-le à la source vehicule (déjà créée),
3. sur un clic droit sur le DataGridView, supprimez des colonnes,
4. transformez numImmaVehicule en ComboBox, que nous lions à la source bdgSvehicule.



chargez les données



Le bouton enregistrer met à jour la base

### Le formulaire Leçon/Valider : version 2 (pour aller un peu plus loin)

Il est dommage que nous identifiions l’élève par son id, qui n’est pas très parlant, et par ses nom et prénom. Par ailleurs, nous allons un peu modifier le formulaire pour ne proposer dans le DataGridView que les leçons non encore validées.

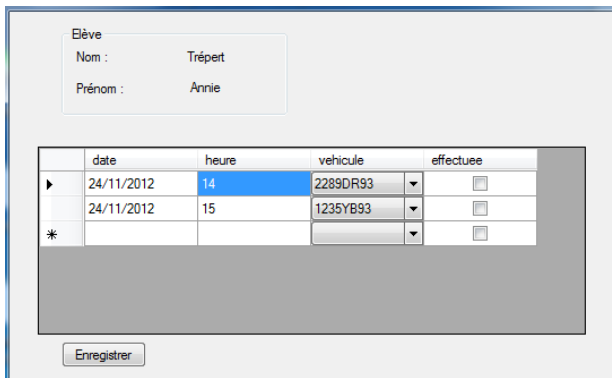
Nous sommes confrontée à une difficulté liée à la conception des DataGridView, ceux-ci ne savent/peuvent pas se lier/binder (dans les deux sens lecture/écriture) à une « jointure », ce qui serait le cas dans l’hypothèse de faire apparaître le nom de l’élève dans le DataGridView. Une solution serait de modifier le comportement initial d’un DataGridView en le spécialisant. Solution que nous écartons.

Nous aurions pu aussi intervenir au niveau du modèle de classe et « migrer » des propriétés de la classe eleve vers la classe lecon, le nom et le prénom de l’élève aurait été ainsi directement accessibles.

Nous procèderons autrement en faisant apparaître dans un autre composant les noms et prénoms des élèves.

Concernant la seconde modification (ne faire apparaître que certaines leçons, celles qui ne sont pas validées), il existe une propriété filter dans une source de données qui aurait pu faire l’affaire (effectuee\_OuiNon = false) mais la documentation MSDN est très incomplète sur cette question.

Donc, nous voulons obtenir un formulaire qui ressemble à ceci :

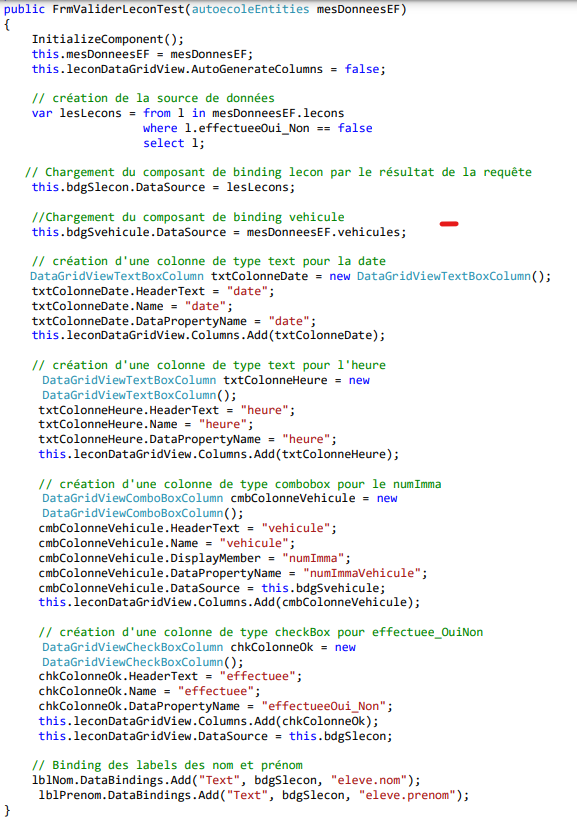


Seules les leçons non validées sont visibles, ainsi que le nom et prénom de l’élève.

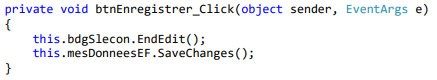
Créez une nouvelle source de données, à partir d’une requête linq.

Remarque : le type de var est IQueryable, ce type peut être bindé (dans les deux sens) ou pas !! ça dépend de la requête : si la requête ne fait que filtrer les lignes (select \* from uneTable where, en SQL), la source de données est bindable dans les deux sens ; si par contre, nous réduisons les colonnes ou faisons une jointure, la source est bindable dans un seul sens (pas de mode edit du composant).

Créez les colonnes du DataGridView par le code, ce qui donne au final :

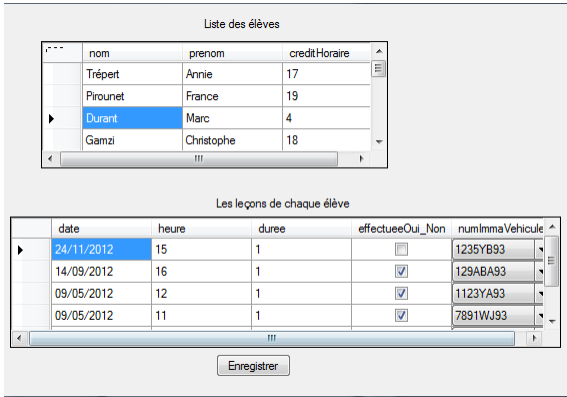


Pour le bouton, nous avons :

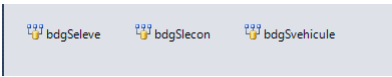


### Un formulaire visualisant les leçons d’un élève : deux DataGridView liés

Nous voulons obtenir les leçons des élèves à partir d’un DataGridView d’élèves :

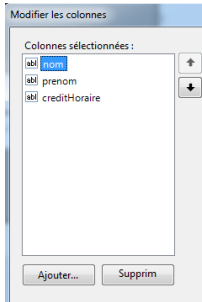


Déposez 3 sources de données :

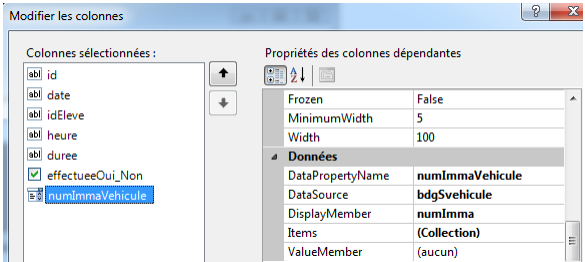


Ces sources sont configurées (propriété dataSource) pour qu’elles pointent vers les classes eleve, lecon et vehicule.

Déposez un DataGridView lié à la source bdgSeleve (propriété dataSource)  
Supprimez les colonnes inutiles :

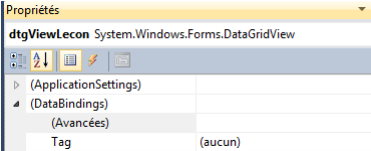


Déposez un second DataGridView lié à la source bdgSlecon,  
modifiez ses colonnes  
remplaçez la colonne texte par un ComboBoxColumn pour le numéro d’immatriculation (cf plus haut) :

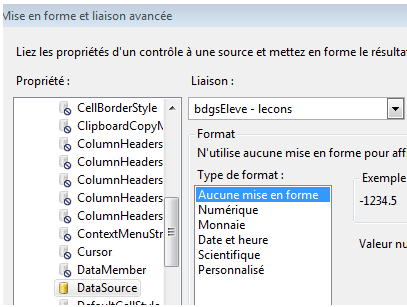


Renseignez la propriété de binding du DataGridView ainsi :

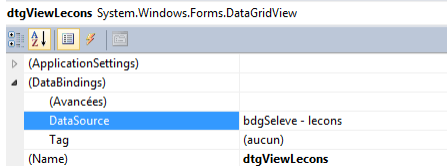
On va chercher la propriété avancée du DataBinding :



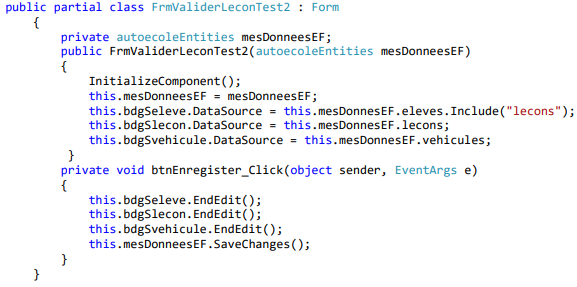
Dans la fenêtre proposée indiquez la DataSource ; ici ce sont les leçons de chaque élève :



Ce qui va produire :



Chargez les sources de données :



La ligne :

this.bdgSeleve.DataSource = this.mesDonneesEF.eleves.Include("lecons");

utilise une méthode Include qui permet de charger pour chaque élève ses leçons. Ce qui n’est pas fait par défaut. EF met en œuvre ce que l’on appelle le lazy loading (chargement paresseux) ; par exemple si on charge en mémoire un objet, **ses** objets associés ne sont pas disponibles ; il faut utiliser explicitement un Include(<nom de la propriété de liaisons>) pour obtenir ses objets liés.

**Annexe 2 : TP Commercial version winform**

**1) Ajout d’un projet WinForm**

- Ouvrir votre projet commercial.

- A partir de l’explorateur de solutions, sélectionner la solution, puis  Click droit/ajouter/nouveau projet

- Indiquer qu’il s’agit d’un projet Winform (application Windows) :

- Indiquer le répertoire du projet (dans le répertoire de votre « solution »)

- Valider

- Votre explorateur de solution vous indique la présence de ce nouveau projet dans la solution :

**TP Commerciaux  version WinForm**

- Sélectionner ce nouveau projet dans l’explorateur de solution

- Faire un click droit sur ce nouveau  projet .

- Sélectionner « définir comme projet de démarrage »

- Click droit  sur ce projet/dépendance du projet, sélectionner le tp Commerciaux (si bien sûr votre projet en mode console s’appelle *tpCommerciaux* :-) ):

- Click droit sur ce projet / Ajouter une référence.

On y est presque, il faudra dans chaque prochain formulaire ajouter : using tpCommerciaux (ceci permet d’accéder directement aux classe sans les préfixer par *tpCommerciaux*). **TP Salarie version WinForm**

**2) Créer un menu**

|  |  |
| --- | --- |
| Déposer un composant  MenuStrip dans le formulaire, puis lancer l’application et vérifier que ces menus et sous menu  apparaissent | |
|  | . |
|  |  |

3) Creation des Formulaires

|  |  |
| --- | --- |
| Saisie d’un commercial  Pour cela :  - ajouter un nouveau formulaire que vous nommerez «  FrmAjoutCommercial »  -déposer les composants ci-dessous | |
|  | **Label (lbl)**  **TextBox(txt)**  -txtNom  -txtPrenom  **ListBox (lst)**  -lstPuissance (prés rempli de chiffres)   Depart  5 à 25  **GroupBox  (grp)**  - grpCategorie  **RadioButton (rdB)**   3 radioButton  -rdBA  -rdBB  -rdBC |

|  |  |
| --- | --- |
| Saisie d’une note de frais  Pour cela :  - ajouter un nouveau formulaire que vous nommerez «  FrmAjoutNote »  -déposer les composants ci-dessous | |
|  | **ListBox (lst)**  -lstCommerciaux  **DateTimePicker**  -datTP  **GroupBox  (grp)            TextBox(txt)**  - grBxkm                          -rdBtTransport                                            -rdBtRepas  **-**rdBtNuite  **GroupBox  (grp)            TextBox(txt)**  -grBxKm                          - txtKm  **GroupBox  (grp)            TextBox(txt)**  - grBxMontant-txtNom  **GroupBox  (grp)            TextBox(txt)**  - grBxkm                         -rdBt1                                           -rdBt2                                           -rdBt3 |

|  |  |
| --- | --- |
| Voir les notes de frais d’un commercial  Pour cela :  - ajouter un nouveau formulaire que vous nommerez «  FrmVoirNotes »  -déposer les composants ci-dessous | |
|  | **listBox (lst)**  -lstCommerciaux    listBox  -lstNotes |

4) CHARGEMENT DES FORMULAIRES

|  |  |
| --- | --- |
| Chargement  des formulaires  En cliquant sur le Sous-menu une procédure s’ouvre : c’est l’endroit ou il faut écrire le code qui sera appelé lorsque l’utilisateur cliquera sur le sous menu, cela permet de désérialisé un fichier ou sérialisé un fichier et de charger les formulaires suivant le sous menu.  Il faut écrire le code qui va permettre d’ouvrir les formulaires  **et fournir les informations  au formulaire voir les notes pour qu’il puisse afficher les commerciaux et les notes de frais pour un commercial selectionné ,** c’est à dire l’objet *leService* déjà construit.  Pour cela, on va utiliser un second constructeur de la classe FrmBulletins :  Le code est donné pour chaque sous menu | |
|  | Le sous menu « ouvrir »  Ouvre une boite de dialogue qui permet de de récupérer le fichier serialisé  Contenant les commerciaux.  Code à inserer dans la procedure  «  ouvrir »  private void ouvrirToolStripMenuItem\_Click(object sender, EventArgs e)  {    try    {      this.openFileDialog1.ShowDialog();      nomFichier=this.openFileDialog1.FileName;      this.leService=PersisteServiceCommercial.charge(nomFichier);    }    catch (Exception ex)    {       MessageBox.Show(ex.Message);    }          }  *Attention !*  *le fichier sérialisé doit être placé dans le repertoire Bin du projet*  *Vous devez créer deux attributs pour récuperer le nom du fichier ainsi que le service* |
|  | Le sous menu « Enregistrer »  Permet la sérialisation des objets.  Code à inserer dans la procedure  «  enregistrer »  private void enregistrerToolStripMenuItem\_Click(object sender, EventArgs e)  {  try    {       PersisteServiceCommercial.sauve(this.leService,nomFichier);    }    catch (Exception ex)    {      MessageBox.Show(ex.Message);    }  } |
|  | Code à inserer dans la procedure « ajouter »  private void ajouterToolStripMenuItem\_Click(object sender, EventArgs e)  {     FrmAjoutCommercial f = new FrmAjoutCommercial(leService);     f.Show();  } |
|  | Code à inserer dans la procedure « ajouterUneNote »  private void ajouterUneNoteDeFraisToolStripMenuItem\_Click(object sender, EventArgs e)  {     FrmAjoutNote f = new FrmAjoutNote(leService);     f.Show();  } |
|  | Chargement du formulaire voir les notes de frais  Code à inserer dans la procedure « ajouter »  private void voirLesNotesToolStripMenuItem\_Click(object sender, EventArgs e)  {    FrmVoirNotes f = new FrmVoirNotes(leService);    f.Show();  } |

**5) Chargement des données**

**1) Ajouter un commercial**

**cas d’utilisation**

Cas :d’utilisation ajout d’un commercial

1- L’utilisateur demande l’ajout d’un commercial

2- Le système retourne le formulaire correspondant

3- L’utilisateur sélectionne une puissance parmi les puissances de la liste  et une catégorie de véhicule

4- Le système enregistre ce nouveau commercial

L’interface peut ressembler à ceci :

public FrmAjoutCommercial(ServiceCommercial leService)

private ServiceCommercial leService;

{

      InitializeComponent();

      this.leService = leService;

}

**// Le formulaire doit s’ouvrir avec la *listBox* chargé avec les numéros de puissances**

private void ajoutCommercial\_Load(object sender, EventArgs e)

{

     for (int i = 5; i < 25; i++)

     this.lstPuissance.Items.Add(i);

}

**// Pour enregistrer ce nouveau commercial, on écrit le code dans l’événement click du bouton :**

private void btnValider\_Click(object sender, EventArgs e)

{

     try

     {

         int puissance = Convert.ToInt32(this.lstPuissance.SelectedItem.ToString());

         char c = 'A';

         if (this.rdBtB.Checked)

                c = 'B';

         if (this.rdBtC.Checked)

                c = 'C';

         Commercial cm = new Commercial(this.txtNom.Text, this.txtPrenom.Text, puissance, c);

         leService.ajouterCommercial(cm);

            }

            catch (Exception ex)

            {

                MessageBox.Show(ex.Message);

            }

        }

**// Pour Fermer Le Formulaire**

        private void btnFermer\_Click(object sender, EventArgs e)

        {

            this.Close();

        }

**2) Ajouter une note de frais**

**cas d’utilisation**

Cas :d’utilisation ajout une note de frais

1- L’utilisateur demande l’ajout d’une note de frais

2- Le système retourne le formulaire avec la liste de tous les commerciaux

3- L’utilisateur sélectionne un commercial, une date et type de frais

4- Le système enregistre cette nouvelle note de frais

Alternatif

3.a l’utilisateur sélectionne le type de frais « transport »

       Le système affiche le nombre de km à saisir

3.b l’utilisateur sélectionne le type de frais « repas »

       Le système affiche le montant à saisir

3.c l’utilisateur sélectionne le type de frais « nuite »

       Le système affiche le montant et la région  à saisir

L’interface peut ressembler à ceci :

Pour que cet appel soit possible, il faut ajouter un attribut privé *leService*  de type ServiceCommercial

 dans la classe FrmAjoutNote et ajouter un second constructeur qui valorise ce champ.

public FrmAjoutNote () // *constructeur déjà présent*

{

     InitializeComponent();

}

// attribut privé

private ServiceCommercial leService;

private char type;

public FrmAjoutNote(ServiceCommercial leService) // modification du constructeur

{

     InitializeComponent();

     this.leService = leService;                // valorisation du champ

}

**// chargement des commerciaux dans la listBox**

private void FrmAjoutNote\_Load(object sender, EventArgs e)

{

     for (int i = 0; i < leService.nbCommerciaux(); i++)

      {

         Commercial c = leService.getCommercial(i);

         string s = c.getNom() + "   " + c.getPrenom();

         this.lstCommerciaux.Items.Add(s);

      }

}

// En function du type des notes de frais aon attribute une valeur a l’attribut type

private void rdBTransport\_CheckedChanged(object sender, EventArgs e)

{

     this.grBxKm.Visible = true;

     this.grBxMontant.Visible  = false;

     this.grBxZone.Visible = false;

     this.type = 'T';

}

private void rdBRepas\_CheckedChanged(object sender, EventArgs e)

{

    this.grBxKm.Visible = false;

    this.grBxMontant.Visible  = true;

    this.grBxZone.Visible = false;

    this.type = 'R';

}

private void rdBNuite\_CheckedChanged(object sender, EventArgs e)

{

   this.grBxKm.Visible = false;

   this.grBxMontant.Visible  = true;

   this.grBxZone.Visible  = true;

   this.type = 'N';

}

**// Pour enregistrer cette nouvelle note, on écrit le code dans l’événement click du bouton :**

private void btnValider\_Click(object sender, EventArgs e)

{

    try

    {

**// On récupére le nom et le prenom dans une vhaine de caracteres**

       string s = this.lstCommerciaux.SelectedItem.ToString();

// la variable tc est le séparateur

       char[] tc ={ ' ' };

// Slipt permet de découper une chaine de caractères en respectant le séparateur

       string[] t = s.Split(tc);

       string nom = t[0];

       string prenom = t[t.Length - 1];

       DateTime dt = this.dateTP.Value.Date;

       Commercial c = leService.getCommercial(nom, prenom);

       char region = '1';

       if (this.rdBt2.Checked)

          region = '2';

       if (this.rdBt3.Checked)

          region = '3';

          switch (this.type)

          {

             case 'T': c.ajouterNote(dt, Convert.ToInt32(this.txtKm.Text)); break;

             case 'R': c.ajouterNote(dt, Convert.ToDouble(this.txtMontant.Text)); break;

      case 'N':c.ajouterNote(dt, Convert.ToDouble(this.txtMontant.Text), region); break;

           }

   }

   catch (Exception ex)

   {

        MessageBox.Show(ex.Message);

    }

}

// Pour fermer le formulaire

private void btnFermer\_Click(object sender, EventArgs e)

{

     this.Close();

}

**2) Voir les notes de frais**

**cas d’utilisation**

Cas : d’utilisation Voir les notes de frais

1- L’utilisateur demande de voir les notes de frais

2- Le système retourne le formulaire avec la liste de tous les commerciaux

3- L’utilisateur sélectionne un commercial

4- Le système retourne la liste de toutes les notes de frais correspondant au commercial

Pour que cet appel soit possible, il faut ajouter un attribut privé *leService*  de type ServiceCommercial

 dans la classe FrmVoirNote et modifier le constructeur pour qu’il valorise ce champ.

private ServiceCommercial leService;

public FrmVoirNotes(ServiceCommercial leService) // constructeur modifier

{

   InitializeComponent();

   this.leService = leService;

}

**// chargement des commerciaux**

private void FrmVoirNotes\_Load(object sender, EventArgs e)

{

       for (int i = 0; i < leService.nbCommerciaux(); i++)

       {

           Commercial c = leService.getCommercial(i);

           string s = c.getNom() + "   " + c.getPrenom();

           this.lstCommerciaux.Items.Add(s);

        }

}

**// En function du commercial selectionné, chargement des notes de frais**

private void lstCommerciaux\_SelectedIndexChanged(object sender, EventArgs e)

{

     this.lstNotes.Items.Clear();

     string s = this.lstCommerciaux.SelectedItem.ToString();

     char[] tc ={ ' ' };

     string[] t = s.Split(tc);

     string nom = t[0];

     string prenom = t[t.Length - 1];

     Commercial c = leService.getCommercial(nom, prenom);

     for (int i = 0; i < c.nbNotes(); i++)

     {

         NoteFrais nf = c.getNoteFrais(i);

         string sn = nf.ToString();

         this.lstNotes.Items.Add(sn);

      }

}