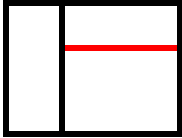
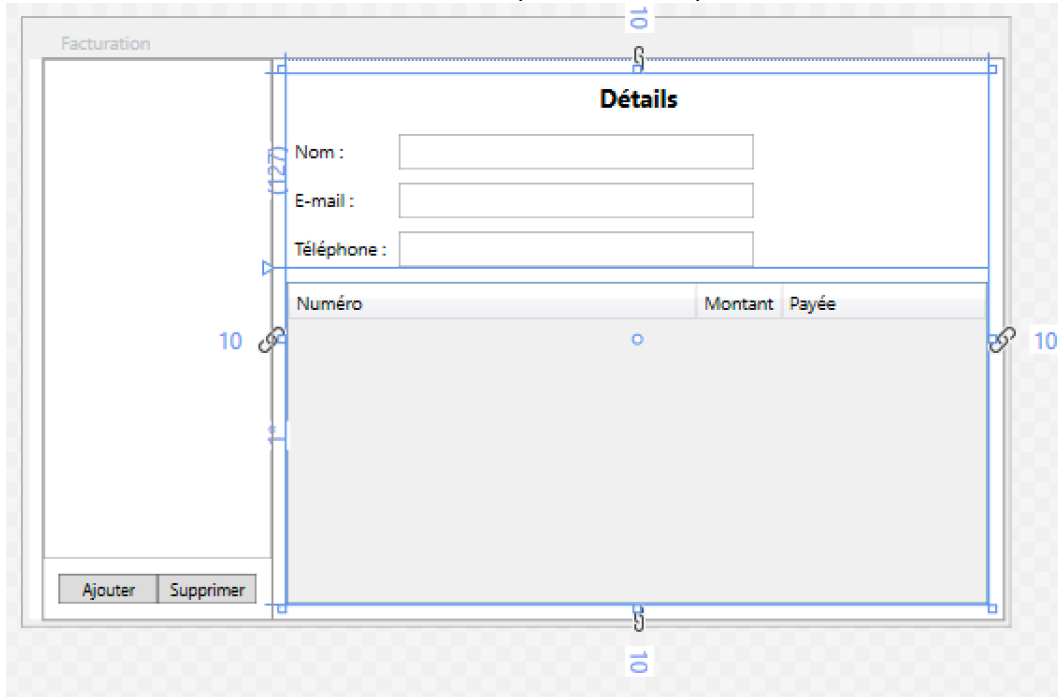




Utilisez 2 grilles (balise Grid) :



Utilisez une listView pour la partie gauche et un DataGridView pour la facture. La listView ainsi que les champs de saisie sont de taille fixe. Les autres composants s'adaptent au redimensionnement.



### 3 Importation et liaison des données

Dans un premier temps nous allons développer notre application avec des données statiques.

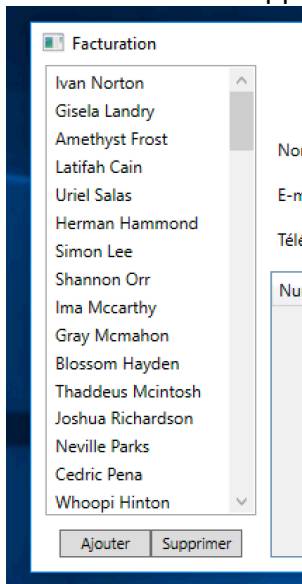
Créez un dossier Model et y importer les classes Client/Facture et Data.

Dans le code C# de l'application (MainWindows.xaml.cs) instanciez une liste de clients en utilisant la classe statique Data.

Donnez un nom à la listView **lvClients** dans la vue.

Liez la liste de clients au composant listView à l'aide de la propriété ItemsSource.

Au lancement de l'application la liste des clients doit s'afficher dans la listView :



Liez le premier client aux champs texte et Datagrid (utilisez les propriétés Text et ItemsSource) :

### Détails

Nom :

E-mail :

Téléphone :

Numero	Montant	Payee	
F2017071	58.8488730205451	<input type="checkbox"/>	
F2017072	96.2257863181763	<input checked="" type="checkbox"/>	
F2017073	39.0675671403611	<input type="checkbox"/>	
F2017074	16.7844837726022	<input checked="" type="checkbox"/>	
		<input type="checkbox"/>	

Sur l'évènement SelectionChanged de la listView mettre à jour les champs.  
Maintenant quand vous cliquez dans la listView le Détails change immédiatement.

## 4 Ajout/suppression de données

Vous allez coder l'insertion d'un nouveau client en C#.

### 4.1 Suppression

Lorsque vous cliquez sur le bouton Supprimer vous retirez de la liste le client sélectionné. N'oubliez pas de mettre à jour le contenu des composants graphiques. Pour la listView utilisez ItemRefresh(). Attention il faut modifier l'évènement SelectionChanged car à la suppression le SelectedIndex passe à -1.

### 4.2 Ajout

Lorsque vous cliquez sur le bouton Ajouter un nouveau client (videz les champs de saisie auparavant) :

#### Détails

Nom :

E-mail :

Téléphone :

Numero	Montant	Payee	
		<input type="checkbox"/>	

Ajoutez un bouton valider caché dans la vue.

A l'aide du code `valider.Visibility = Visibility.Visible;` faites apparaître pour insérer les données saisies dans la liste.

**Pour des questions de simplicité vous ne coderez que l'insertion du nom, e-mail et téléphone.**

N'oubliez pas de rafraichir les composants.

## 5 Binding

Comme vous avez pu le constater le codage behind est très fastidieux. Nous allons mettre en place le principe du Binding qui permet de lier dynamiquement les composants aux données. On va établir une liaison bidirectionnelle à l'aide d'une propriété de dépendance s'il s'agit d'un objet et une propriété publique de type Observable s'il s'agit d'une liste.

### 5.1 Préparation

Supprimez toutes vos propriétés et objets dans MainWindow. Supprimez les événements ainsi que les méthodes associées. Ne conserver que `InitializeComponent()` dans MainWindow. Supprimez le bouton caché Valider.

Votre code est quasiment vide !

### 5.2 Création des objets et listes

Propriété LesClients :

Créez la liste de type Observable LesClients :

```
public ObservableCollection<Client> LesClients { get; set; }
```

Régénérez la solution et vérifiez que vous n'avez pas d'erreur.

### 5.3 Liaisons

Liez le conteneur de gauche (ListView) avec la collection Observable :

```
ItemsSource="{Binding LesClients, RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type local:MainWindow}}}"
```

Liez le conteneur de droite (grille) avec la dépendance :

```
DataContext="{Binding SelectedItem, ElementName=lvClients}"
```

Liez alors chacun des composants :

Pour le Datagrid :

```
ItemsSource="{Binding Factures}"
```

Créez des colonnes (ajouter au DataGrid `AutoGenerateColumns="False"`):

```
<DataGrid.Columns>
    <DataGridTextColumn Header="Numéro" Binding="{Binding Numero}" Width="2*"/>
    <DataGridTextColumn Header="Montant" Binding="{Binding Montant}" Width="Auto"/>
    <DataGridCheckBoxColumn Header="Payée" Binding="{Binding Payee}" Width="*/>
</DataGrid.Columns>
```

Exemple texte du téléphone : `Text="{Binding Telephone}"`

Rq : il est possible de le faire graphiquement

### 5.4 Chargement des données

Dans MainWindow chargez les clients :

```
LesClients = new ObservableCollection<Client>(Data.Load());
```

```
lvClients.SelectedItem = LesClients[0];
```

Lancez l'application. Tout fonctionne y compris la modification !

Facturation

Ivan Norton

Gisela Landry

Amethyst Frost

Latifah Cain

Uriel Salas

Herman Hammond

Simon Lee

Shannon Orr

Ima Mccarthy

Gray Mcmahon

Blossom Hayden

Thaddeus Mcintosh

Joshua Richardson

Neville Parks

Cedric Pena

Whoopi Hinton

Ajouter

Supprimer

Détails

Nom : Uriel Salas

E-mail : Proin@tincidunt.ca

Téléphone : 09 25 64 29 00

Numéro	Montant	Payée
F20170712	33.649465291132	<input checked="" type="checkbox"/>
F20170713	73.2834463023038	<input type="checkbox"/>
F20170714	29.4352962958791	<input checked="" type="checkbox"/>
F20170715	72.2389928476135	<input type="checkbox"/>
		<input type="checkbox"/>

Le montant n'est pas très lisible. Ajoutez le code d'initialisation de la langue locale :  
`this.Language = XmlLanguage.GetLanguage(Thread.CurrentThread.CurrentCulture.IetfLanguageTag);`

Ajoutez un filter d'affichage sur le montant afin de passer en euros :  
`Binding="{Binding Montant, StringFormat=C}"`

Numéro	Montant	Payée
F2017071	78,25 €	
F2017072	6,95 €	
F2017073	123,28 €	

## 6 Ajout de style

Nous allons ajouter des styles afin d'améliorer la lisibilité de l'application. Les factures non payées seront en rouge, les payées en vert.

Ajoutez un style sur les cellules du datagrid (nouvelle ressource à placer avant le premier grid) :

```

<Window.Resources>
<Style TargetType="{x:Type DataGridCell}">
    <Style.Triggers>
        <DataTrigger Binding="{Binding Path=Payee}" Value="False">
            <Setter Property="Background" Value="Red"/>
        </DataTrigger>
        <DataTrigger Binding="{Binding Path=Payee}" Value="True">
            <Setter Property="Background" Value="LightGreen"/>
        </DataTrigger>
    </Style.Triggers>
</Style>
</Window.Resources>

```

Le style s'applique automatiquement :

Numéro	Montant	Payée
F20170730	13,28 €	<input checked="" type="checkbox"/>
F20170731	19,90 €	<input type="checkbox"/>
F20170732	83,84 €	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

## 7 Suppression

La suppression devient extrêmement simple à coder !

```
private void SuppClient_Click(object sender, RoutedEventArgs e)
{
    Client cltASupprimer = (Client)lvClients.SelectedItem;
    LesClients.Remove(cltASupprimer);
}
```

Le binding met à jour automatiquement !

## 8 Validateurs

Nous allons mettre en place des validateurs sur les champs de saisie afin de vérifier qu'ils ne sont pas vides à la modification. Nous allons utiliser les setters dans le modèle ainsi que du binding validé par exception.

Modifiez la classe client (exemple nom) :

```
private string nom;
public string Nom
{
    get { return nom; }
    set
    {
        if (string.IsNullOrEmpty(value))
            throw new ArgumentException("Nom obligatoire");
        nom = value;
    }
}
```

Validez le binding des textBox en utilisant les exceptions :

```
Text="{Binding Nom, ValidatesOnExceptions=True}"
```

Appliquez un template (dans Window.resources) au textBox pour lui indiquer comment se comporter lorsqu'une exception est déclenchée :

```
<ControlTemplate x:Key="txtError">
    <StackPanel Orientation="Horizontal">
        <Border BorderBrush="Red" BorderThickness="1">
            <AdornedElementPlaceholder x:Name="element"/>
        </Border>
        <TextBlock x:Name="tbError" Foreground="Red" Margin="5,0,0,0" FontSize="10pt"
Text="{Binding ElementName=element,
Path=AdornedElement.(Validation.Errors)[0].ErrorContent}"/>
    </StackPanel>
</ControlTemplate>
```

Appliquez la ressource au textBox concerné :

```
<TextBox x:Name="txtNom" Margin="74,40,0,0" HorizontalAlignment="Left" Width="244" Height="23" VerticalAlignment="Top" Validation.ErrorTemplate="{StaticResource txtError}">
```

Faites de même avec les 2 autres TextBox

Testez :

## Détails

Nom :	<input type="text"/>	Nom obligatoire
E-mail :	<input type="text" value="sed.libero@velitduisemper.com"/>	
Téléphone :	<input type="text" value="03 98 32 59 47"/>	

Rq : il existe une autre technique plus générique : Les Validation Rules

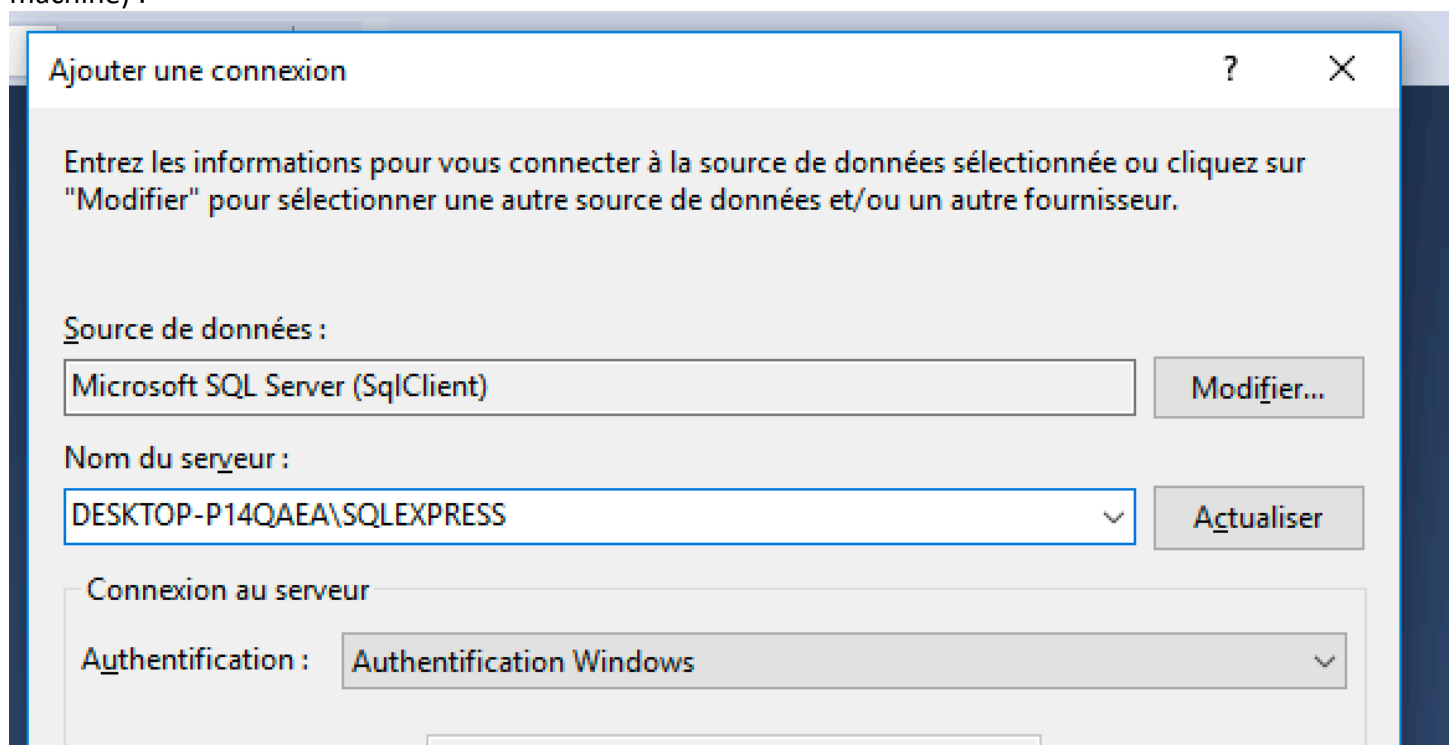
## 9 Base de données, Entity Framework

Dernière étape, nous allons lier nos données avec une base SQL grâce au Framework Entity Framework.

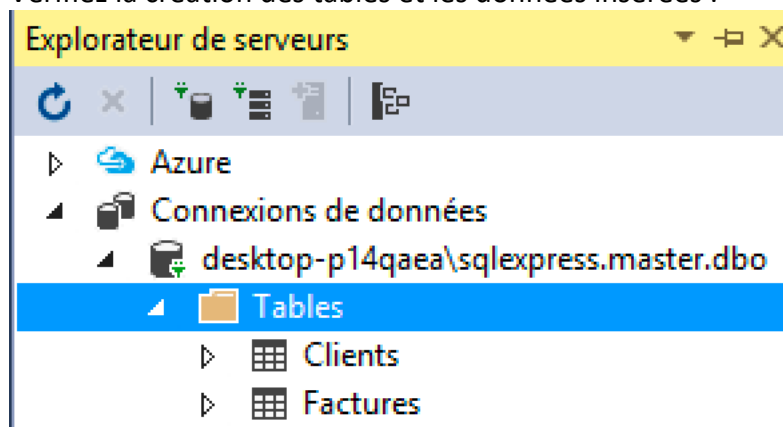
### 9.1 Création de la base de données

Tout d'abord supprimez le répertoire Model contenant les classes métiers.

Créez une base de données locale à l'aide de SQLExpress. Sélectionnez le serveur local (dépend de votre machine) :



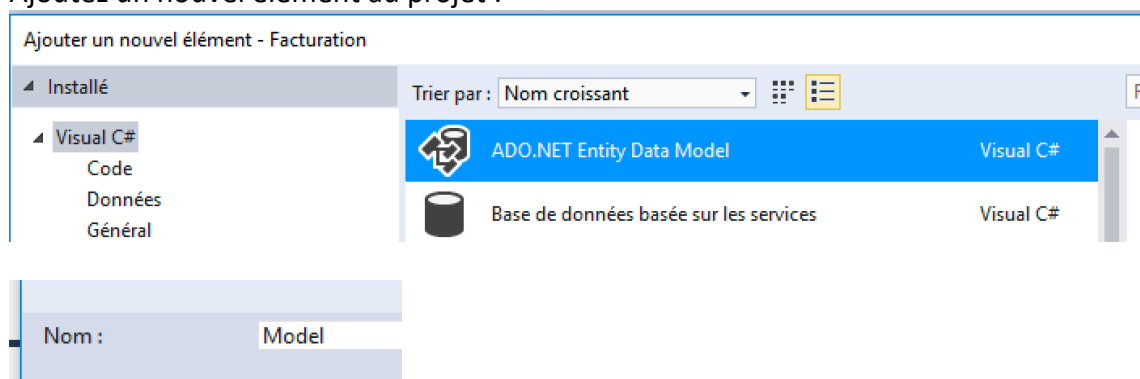
Exécutez le script SQL de création des tables disponible sur le réseau.  
Vérifiez la création des tables et les données insérées :



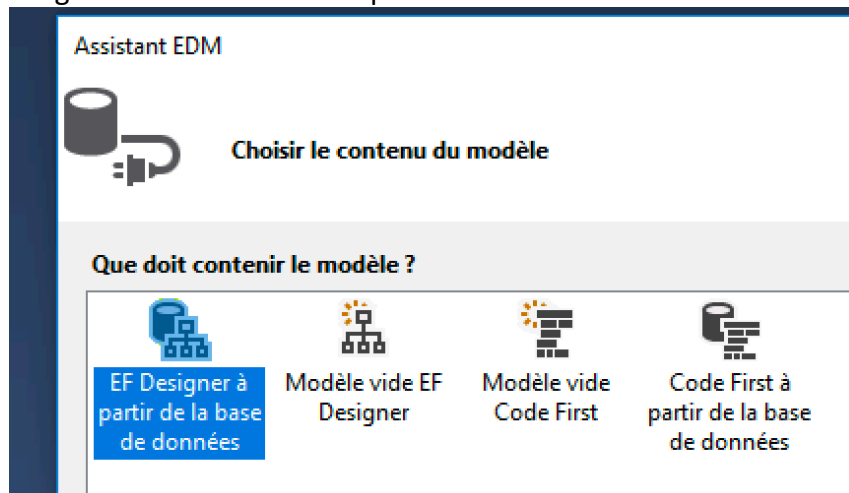
Rq : on pourrait utiliser un serveur SQL dans le cadre d'une utilisation multiutilisateurs.

## 9.2 Liaison de la base de données avec le code métier

Ajoutez un nouvel élément au projet :




On génère le code métier à partir de la base de données :





Assistant EDM

 Choisir votre connexion de données

Quelle connexion de données votre application doit-elle utiliser pour établir une connexion à la base de données ?

desktop-p14qaea\squlexpress.master.dbo ▼ Nouvelle connexion...

Cette chaîne de connexion semble contenir des données sensibles (par exemple, un mot de passe), lesquelles sont indispensables pour établir une connexion à la base de données. Le stockage des données sensibles dans la chaîne de connexion peut entraîner un risque de sécurité. Voulez-vous inclure les données sensibles dans la chaîne de connexion ?

☐ Non, exclure les données sensibles de la chaîne de connexion. Je définirai ces informations dans le code de mon application.

☐ Oui, inclure les données sensibles dans la chaîne de connexion.


Chaîne de connexion :

```
metadata=res://*/Model.csdl|res://*/Model.ssdl|
res://*/Model.msl;provider=System.Data.SqlClient;provider connection string="data source=DESKTOP-
P14QAEA\SQLEXPRESS;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Enregistrer les paramètres de connexion dans App.Config en tant que :

WPFEntities

Assistant EDM


 Choisir votre version

Quelle version d'Entity Framework voulez-vous utiliser ?


☒ Entity Framework 6.x


Bien sélectionner les tables

Assistant EDM

 Choisir vos paramètres

Quels objets de base de données voulez-vous inclure ?

☒  Tables

☒  dbo

Attention cliquez sur Mettre au pluriel :

☒ Mettre au pluriel ou au singulier

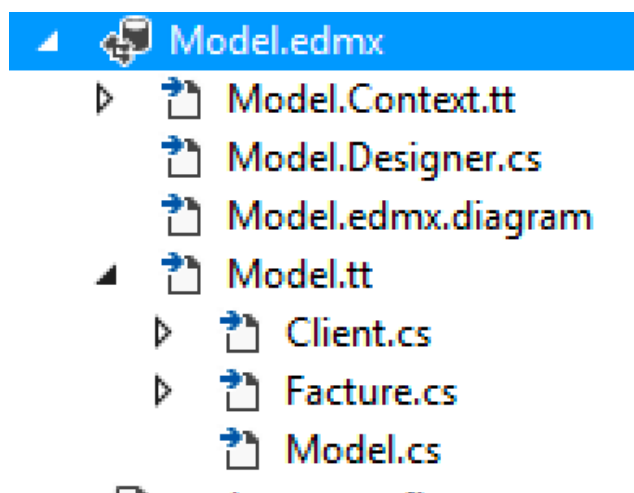
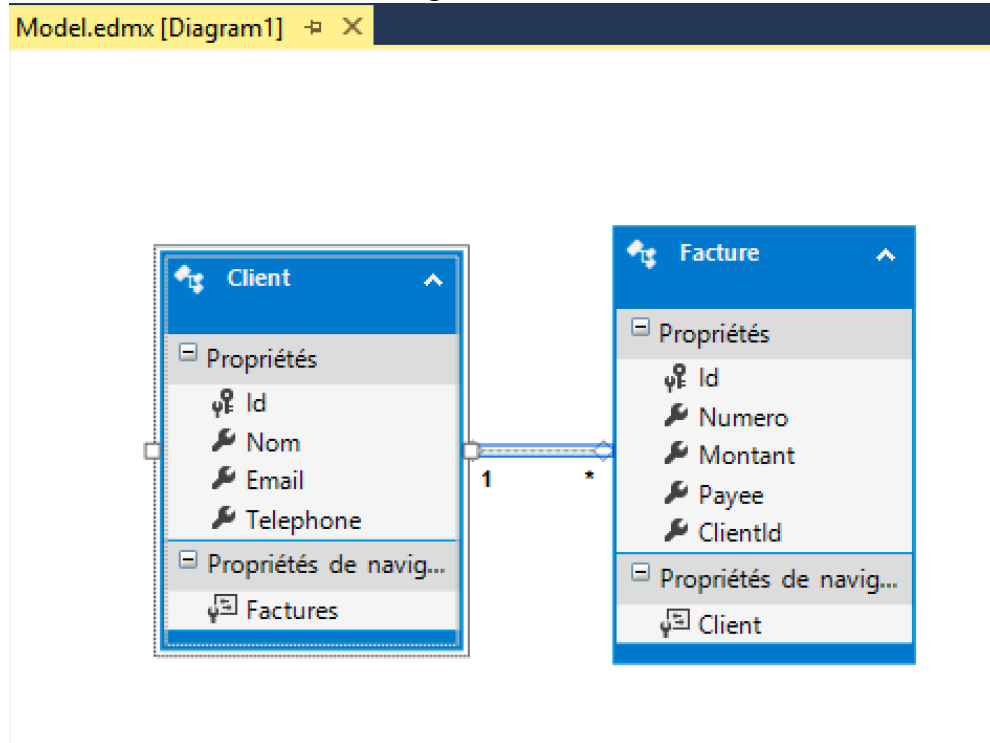
☒ Inclure les colonnes de clés étrangères

☒ Importer les fonctions et les propriétés

Espace de noms du modèle :

ortie :

On obtient alors les classes autogénérées :



Modifiez la classe Client afin de rajouter le méthode ToString() :

```
public override string ToString()
{
    return Nom;
}
```

### 9.3 Accès aux données (Lecture/Ecriture)

On se sert de la classe qui a été autogénérée pour instancier un contexte de données (incluant la connexion). Attention il faut penser à fermer la connexion (ou plutôt libérer le contexte) en utilisant le dispose en cas de fermeture de l'application (Dispose()).

Le code devient de plus en plus simple :

```
public partial class MainWindow : Window, IDisposable
{
    private WPFEntities _context;
    public ObservableCollection<Client> LesClients { get; set; }
    public Client MonClient
    {
        get { return (Client)GetValue(MonClientProperty); }
        set { SetValue(MonClientProperty, value); }
    }

    public static readonly DependencyProperty MonClientProperty =
        DependencyProperty.Register("MonClient", typeof(Client), typeof(MainWindow), new
        PropertyMetadata(null));

    public MainWindow()
    {
        InitializeComponent();
        _context = new WPFEntities();
        LesClients = new ObservableCollection<Client>(_context.Clients.ToList());
        lvClients.SelectedItem = LesClients[0];
        this.Language =
        XmlLanguage.GetLanguage(Thread.CurrentThread.CurrentCulture.IetfLanguageTag);
    }

    public void Dispose()
    {
        if (_context != null)
        {
            _context.Dispose();
            _context = null;
        }
    }

    private void lvClients_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        _context.SaveChanges();
    }
}
```

Bonus : Codez Ajouter/Supprimer un client