REF : SENT-TEC

BDO Sentinel

Documentation Technique Générale

Documentation technique

****

Introduction technique au projet

Version – 2.0

Table des matières

[A. Présentation 4](#_Toc476571181)

[A.1. But du document 4](#_Toc476571182)

[B. Environnement de développement 5](#_Toc476571183)

[B.1. Visual studio 2015 5](#_Toc476571184)

[B.1.a. Install 5](#_Toc476571185)

[B.1.b. Outils complémentaires pour Visual Studio 5](#_Toc476571186)

[B.1.c. Options diverses 6](#_Toc476571187)

[B.2. Team Foundation server 8](#_Toc476571188)

[B.2.a. Connexion au serveur de sources 8](#_Toc476571189)

[B.3. SQL 9](#_Toc476571190)

[B.4. Frameworks et librairies 10](#_Toc476571191)

[B.4.a. Gestion NUGET packages 10](#_Toc476571192)

[B.4.b. Divers 10](#_Toc476571193)

[C. SOLUTION TECHNIQUE 30](#_Toc476571194)

[C.1. Choix effectués 30](#_Toc476571195)

[C.2. Organisation du code 32](#_Toc476571196)

[C.2.a. SHELL 33](#_Toc476571197)

[C.2.b. MODULES 33](#_Toc476571198)

[C.2.c. INFRASTRUCTURE 33](#_Toc476571199)

[C.2.d. SERVICES 34](#_Toc476571200)

[C.2.e. Référencement entre projets 35](#_Toc476571201)

[C.3. PRISM 36](#_Toc476571202)

[C.3.a. Initialisation 36](#_Toc476571203)

[C.3.b. Modularité 37](#_Toc476571204)

[C.3.c. MVVM 37](#_Toc476571205)

[C.3.d. Interface utilisateur et navigation 39](#_Toc476571206)

[C.3.e. Communication 40](#_Toc476571207)

[C.4. Accès aux données 42](#_Toc476571208)

Gestion des versions

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro | Emetteur | Description | Date |
| V1.0 | MBU | Rédaction première version du document | 06/03/2017 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Création du document par Michaël Buisson, le 06/03/2017 08:54

Dernière sauvegarde effectuée par Michael Buisson, le 06/03/2017 08:57

# Présentation

## But du document

Ce document représente une base de travail pour un audit externe de l’architecture logicielle mise en place.

Par audit, nous entendons la validation « théorique » de la solution proposée face aux « Best Practices ».

Nous présentons ici les principes retenus pour la mise en place de notre projet et de son architecture.

Nous n’entrerons pas dans les détails « métiers » mais des extraits de code seront repris pour illustrer les méthodes appliquées pour l’implémentation de la solution.

**Ce que ce document n’est pas :**

Un document détaillé reprenant la description des besoins métiers et amenant à la décision de certains choix techniques.

Il n’est pas non plus un document complet sur l’architecture logicielle et matérielle appliquée à chaque processus.

# Environnement de développement

## Visual studio 2015

### Install

Installer la version « Professional Update 3». Les sources sont disponibles sur :

\\Dglux3\dev-appli\Outils\_développement\_environnement\_visual\_studio\VS2015 Pro

… procédure TBC …

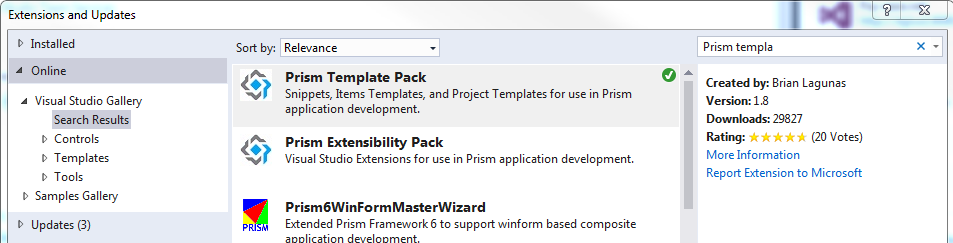
### Outils complémentaires pour Visual Studio

La liste suivante est non exhaustive et peut être complétée selon les besoins.

Les sources d’installation sont soit disponibles sur nos serveurs (outils sous licence ou déjà téléchargés) soit récupérables via le menu « Tools\Extensions and Updates… » de Visual Studio.

L’interface de ce menu est composée de 3 volets :

* Gauche : choix de la recherche (produits installés, mises à jour possibles, produits non installés disponibles online).
* Centre : produits disponibles selon les critères de recherche
* Droite : possibilité de recherche par nom et détail du produit sélectionné



#### WinMerge

Il s’agit d’un outil utilisé pour la comparaison et fusionner des sources, plus convivial que celui proposé par défaut par VS : [\\Dglux3\dev-appli\Outils\_développement\_environnement\_visual\_studio\WinMerge\_et\_plugins](file:///\\Dglux3\dev-appli\Outils_développement_environnement_visual_studio\WinMerge_et_plugins)

La note pour son intégration à Visual Studio se trouve dans le même répertoire.

#### Visual Studio Installer Projects (non nécessaire pour Sentinel)

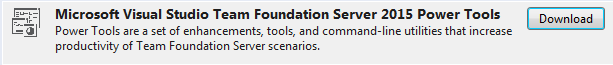
Permet l’ouverture de certains types de projets qui ne sont plus supportés avec les nouvelles versions de Visual Studio.



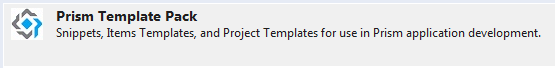
#### Team Foundation Server Power Tools 2015

Extension des possibilités offertes par TFS pour la gestion des sources via Visual Studio.

***Attention : Il n’est possible d’installer qu’une seule version sur un poste (ex : VS2010 OU VS2015)***



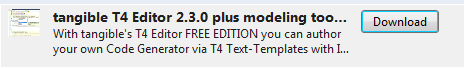
#### Prism Template Pack



#### T4 editor

Améliore l’expérience d’édition des templates T4.

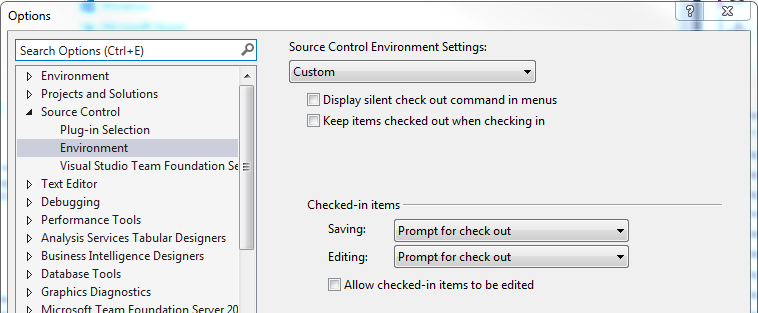
Attention : taper le mot clé ‘Tangible’ pour obtenir la bonne version dans le résultat.



### Options diverses

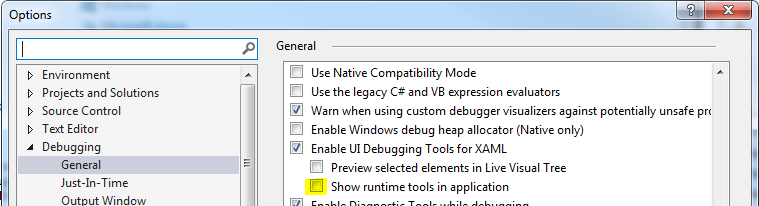
Les options sont accessibles via le menu Tools\Options

#### Contrôle des sources TFS



#### Barre de menu « debug »

Il faut décocher l’option suivante:



Pour enlever les liens ci-dessous dans la barre de menu de l’application



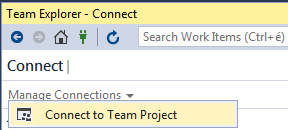
## Team Foundation server

### Connexion au serveur de sources

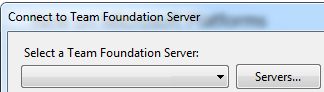
Si la connexion vers TFS n’a pas encore été configurée :

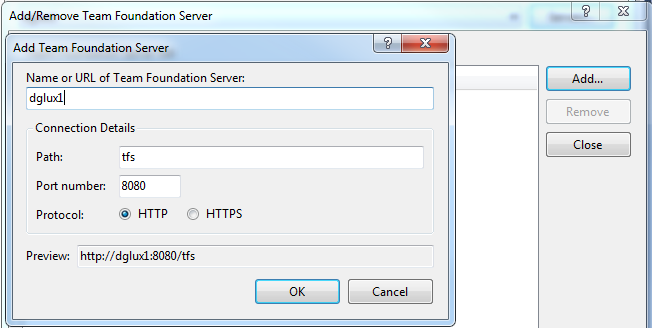


Dans la fenêtre « Team Explorer »

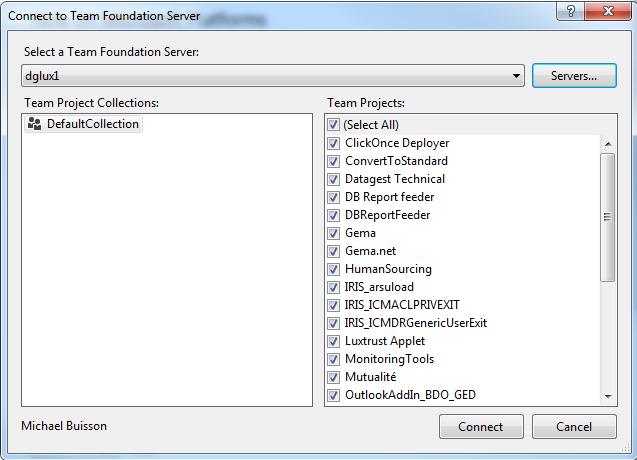


Dans la fenêtre suivante choisir « Servers… »

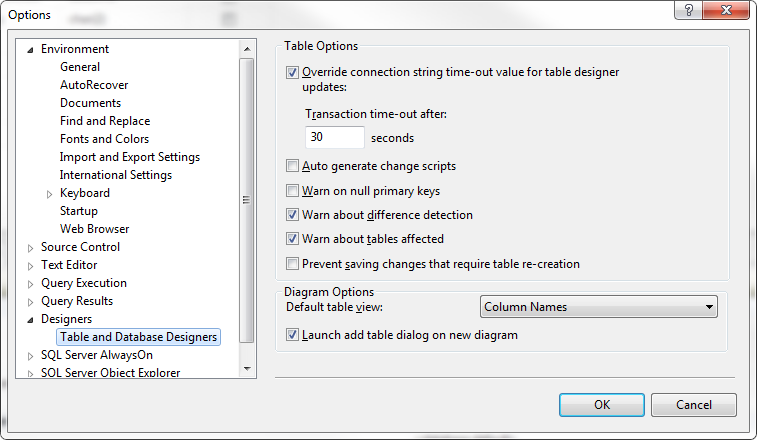




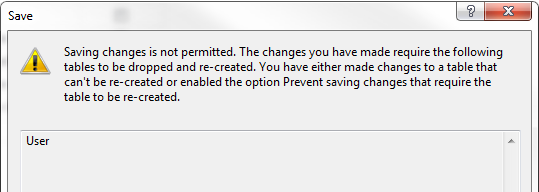
Choisir la liste des projets qui seront visibles dans Visual Studio



## SQL



Pour autoriser la modification d’une table en mode design et ne plus avoir le message suivant :



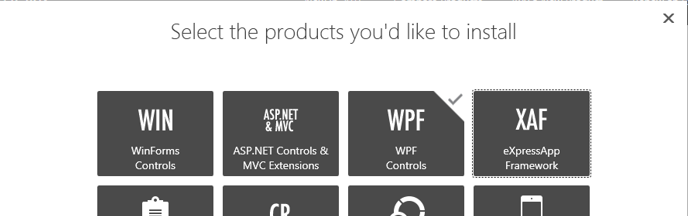
## Frameworks et librairies

### Gestion NUGET packages

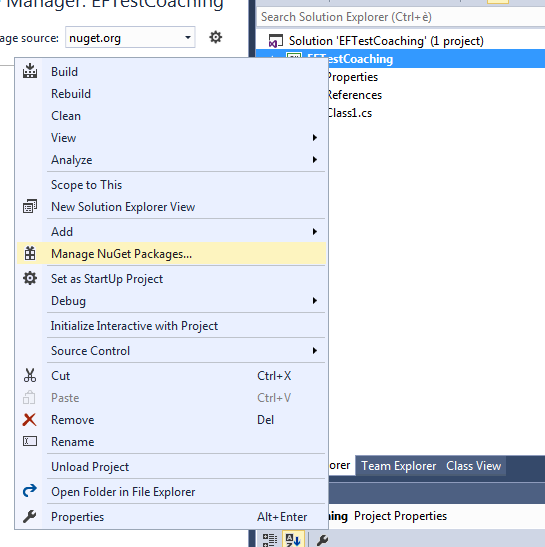
### Divers

#### DevExpress

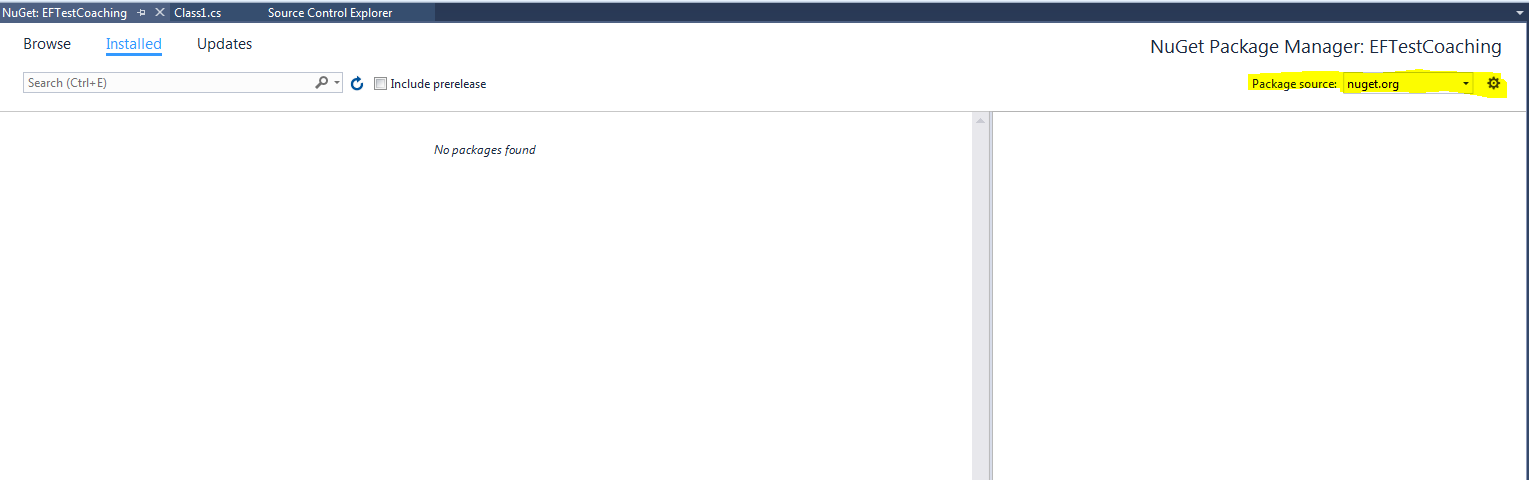
Version 16.1.2



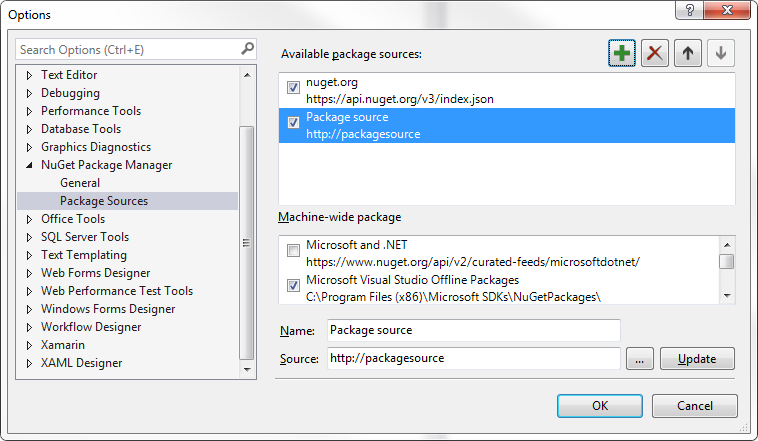
How to change NuGet Package Source :



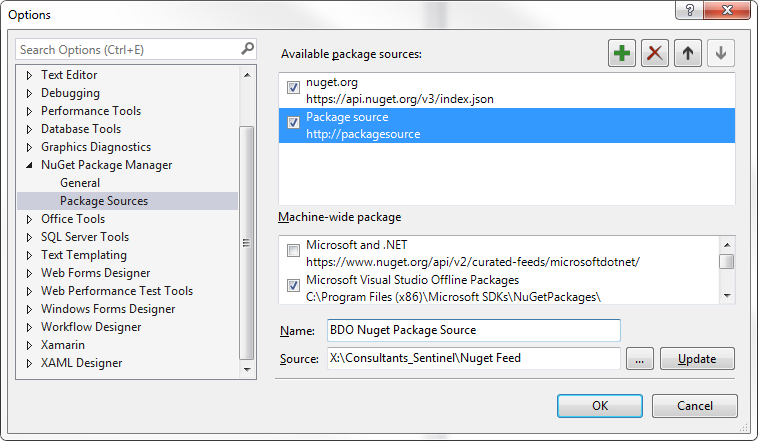
On sélectionne Package Source (la petite roue)



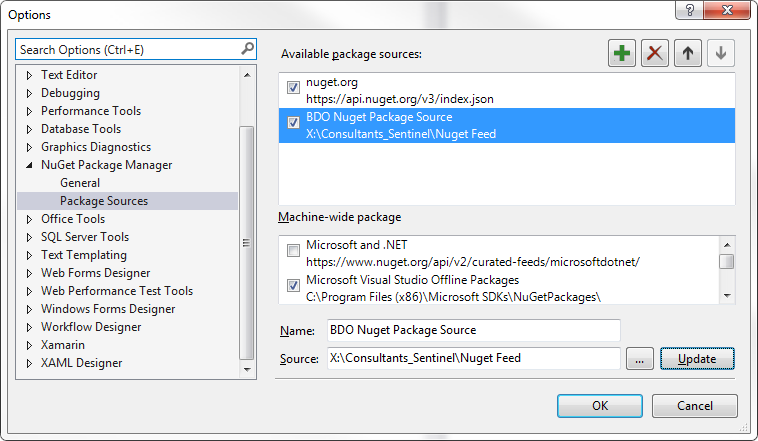
Une fenêtre appraît et on clique sur le +



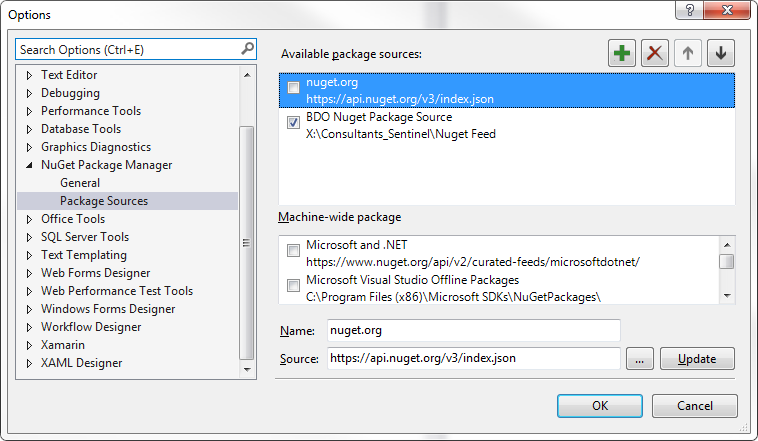
Ensuite on modifie les valeurs pour l’élément Package Source

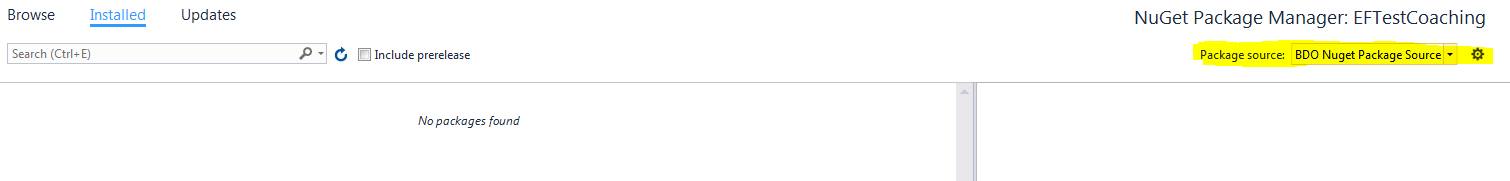


Appuyer sur Update avant de Faire Ok



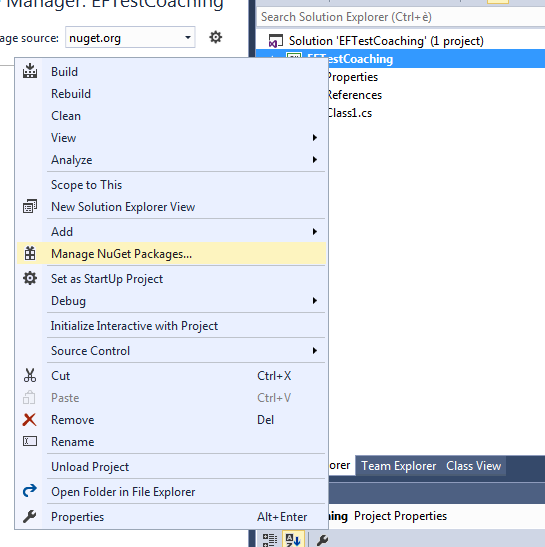
Pour s’assurer de ne prendre les packages que dans le répertoire BDO, il faut décocher les autres options



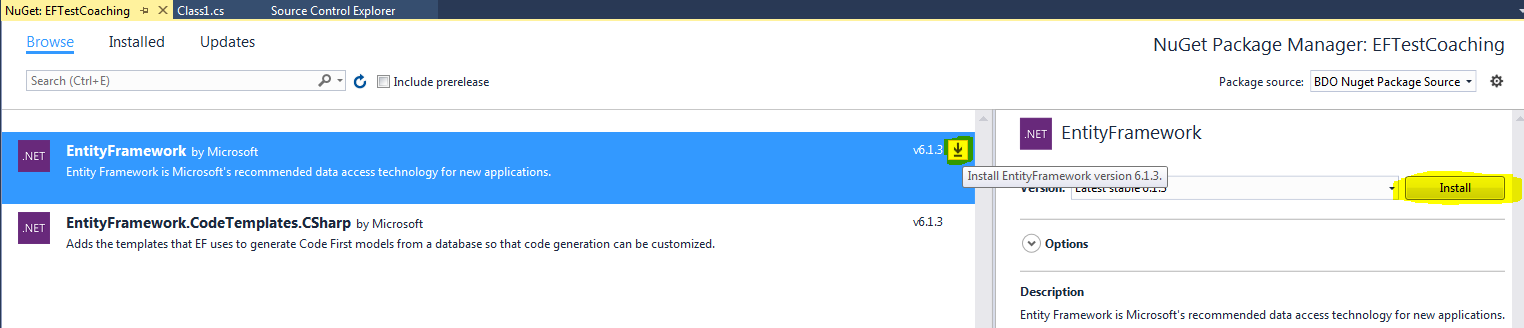


Installation de Entity Framework v6.1.3

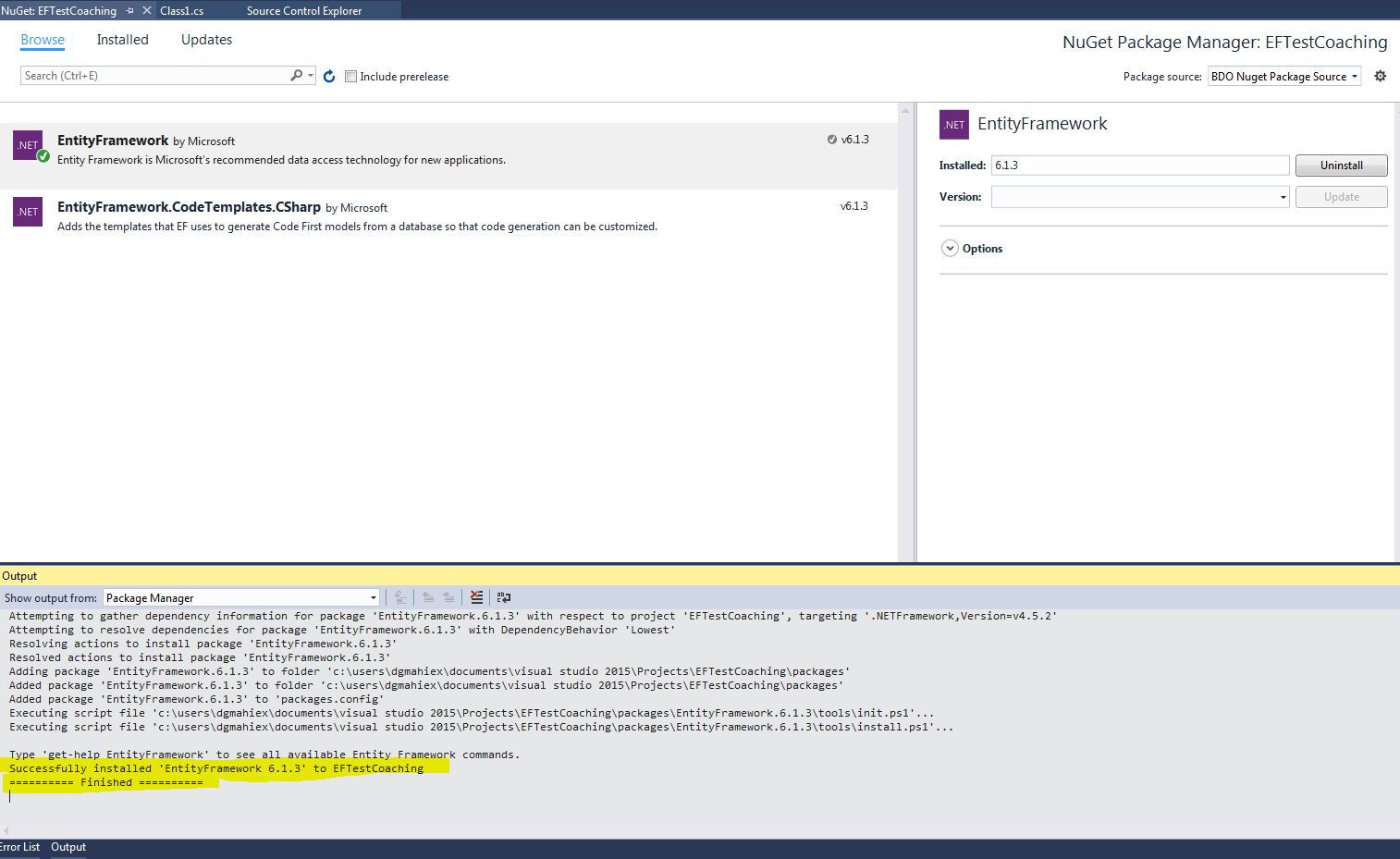
Aller sur Manage NuGetPackage et sélectionner Browse



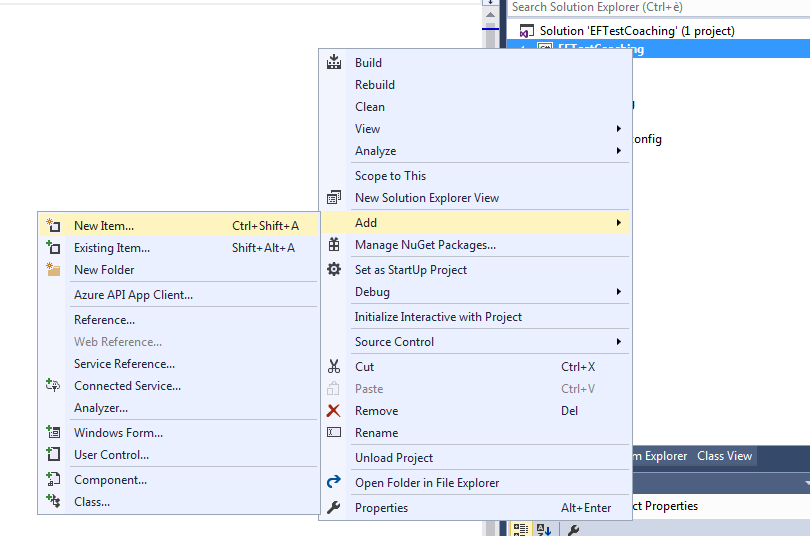
Ensuite Sélectionner EntityFramework et cliquer sur Install ou la flèche

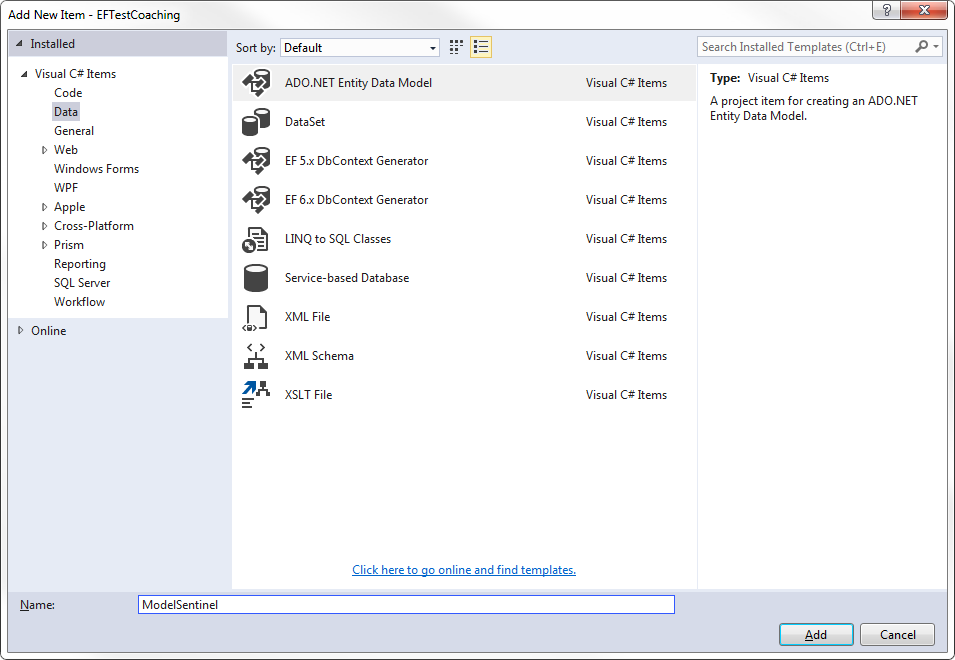


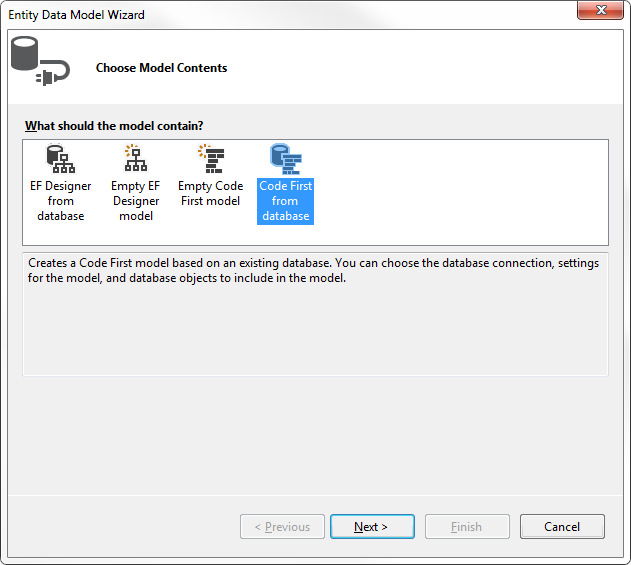
On clique sur « I Accept » et la fenêtre suivante apparaît avec le message comme quoi ça a été bien installé

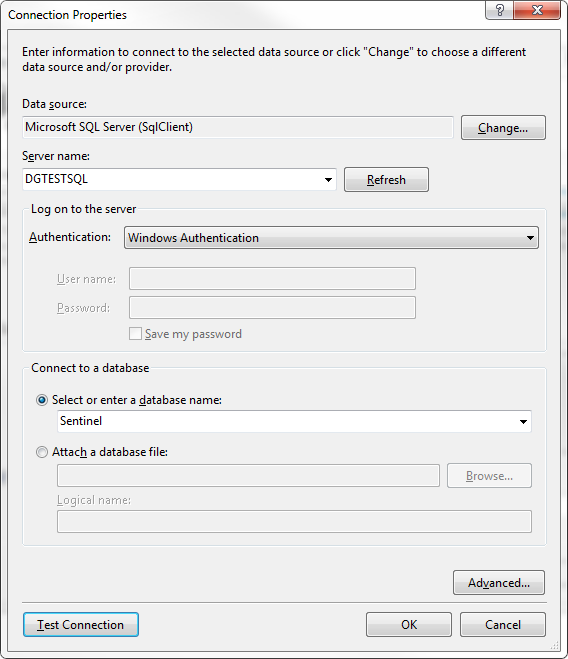


Comment générer les entités avec EF 6

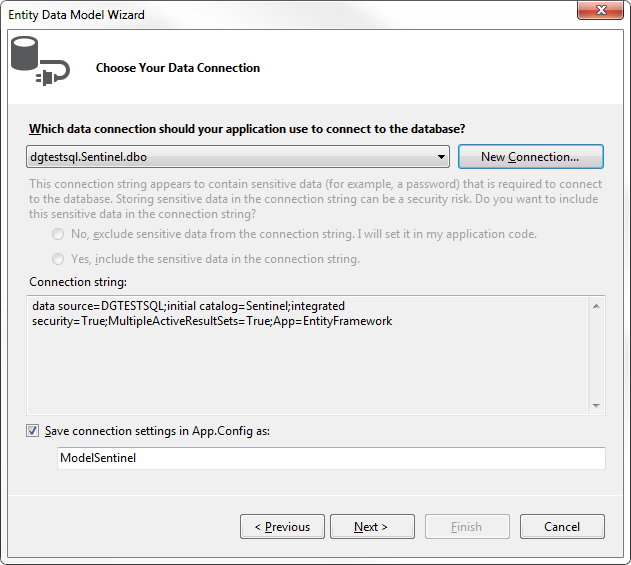






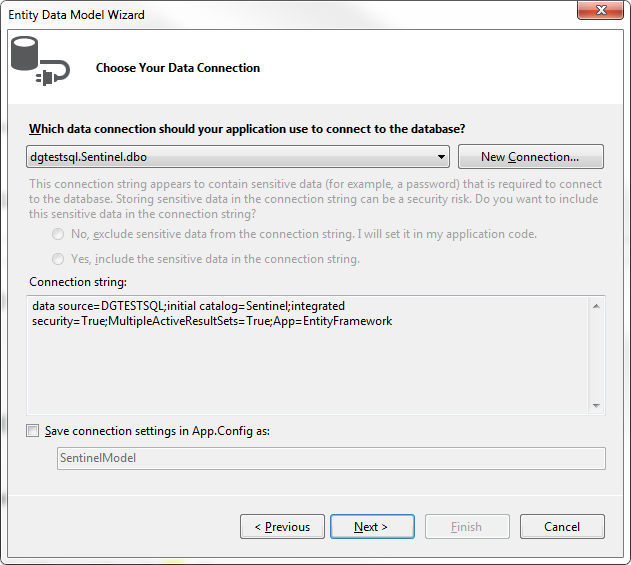


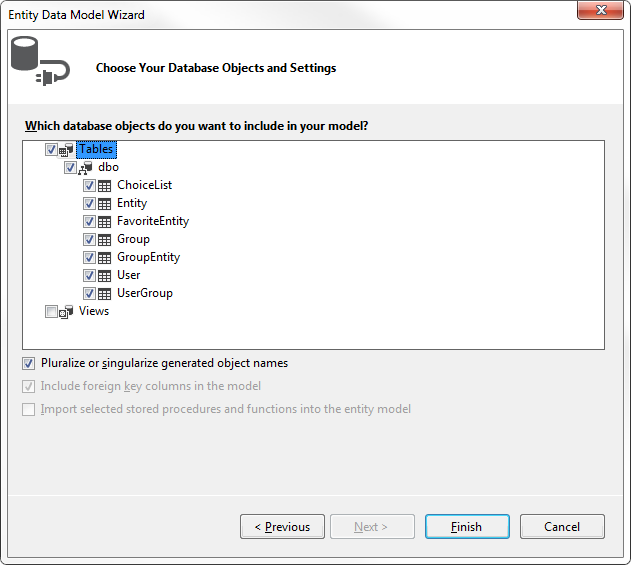
Clique sur Ok



A faire seulement lors de la première génération

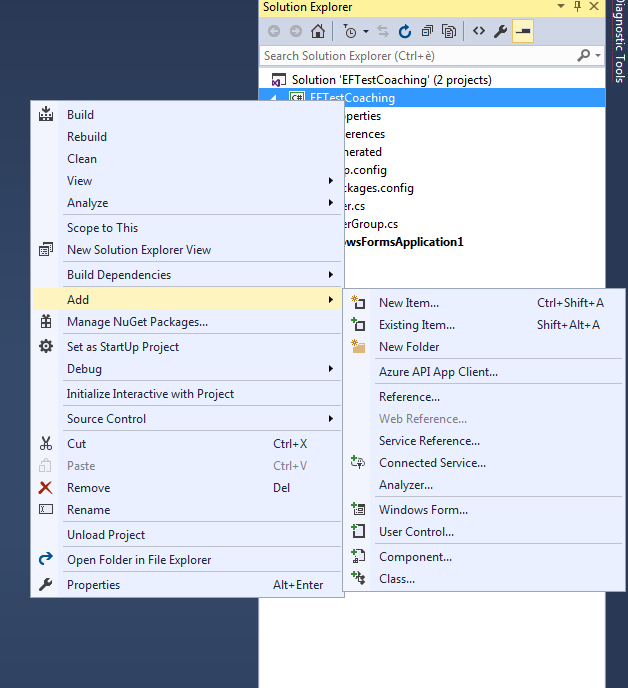
Ensuite suite la fenêtre suivante

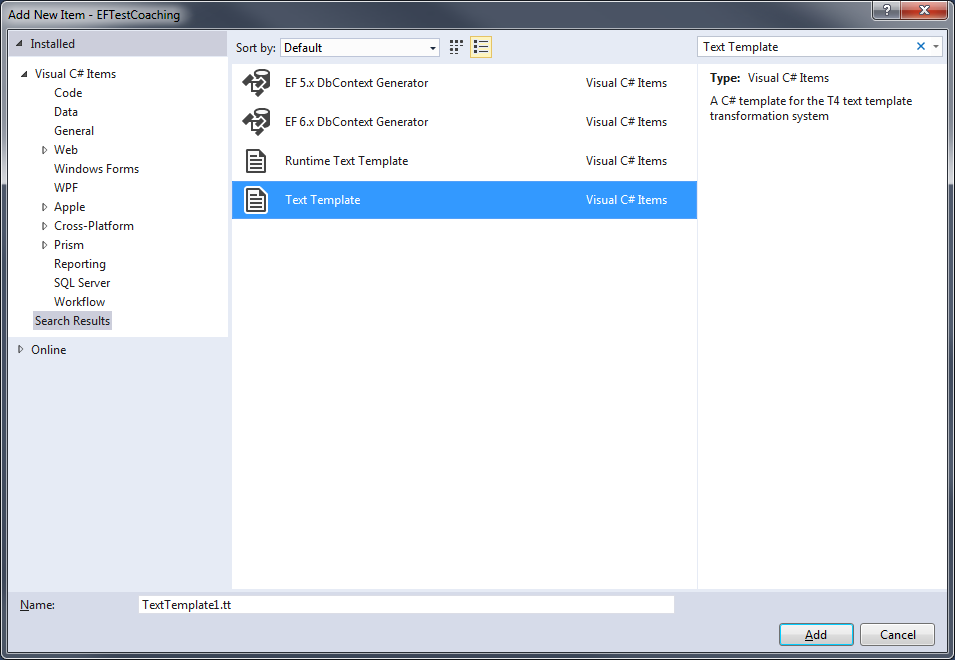




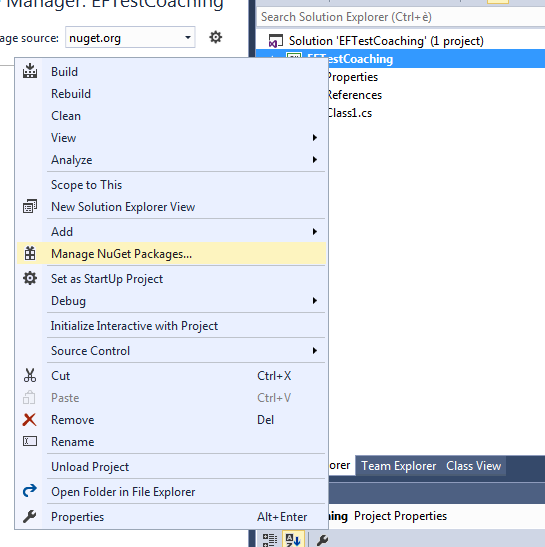
Bien coché Pluralize …

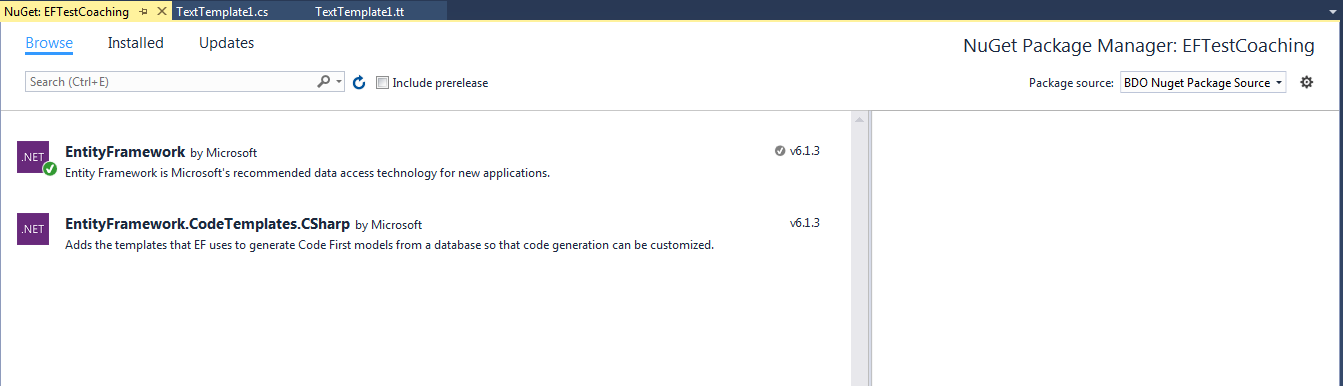
Add T4 template file

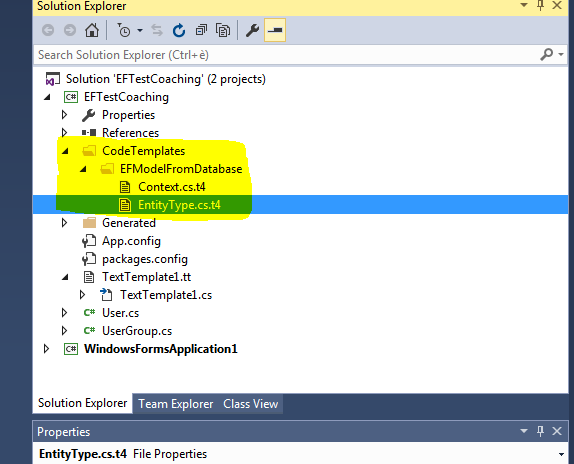




Add NuGet Package Code Template pour créer le template de génération





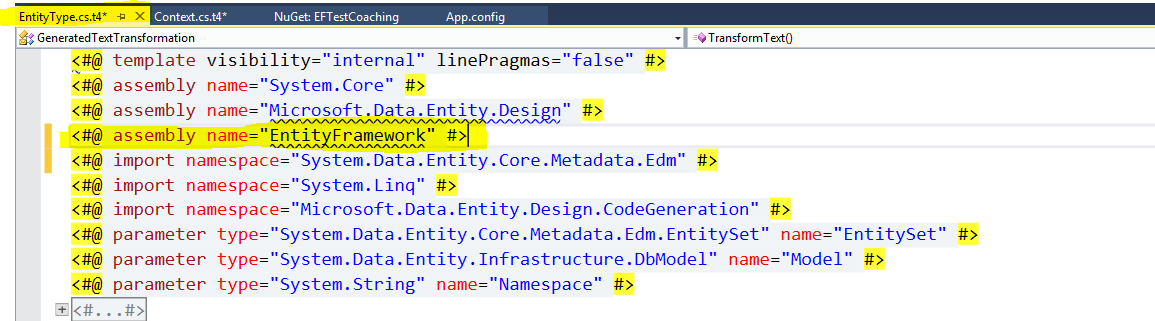


Ne pas oublier de retirer les lignes d2crites ci-dessous dans les images

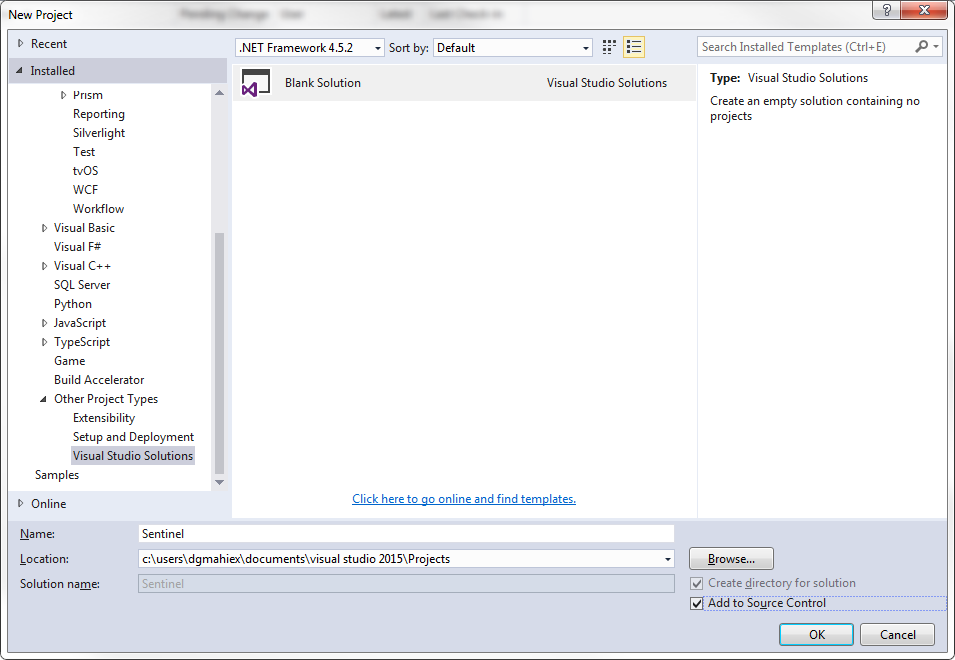
Fichier Context.cs.t4



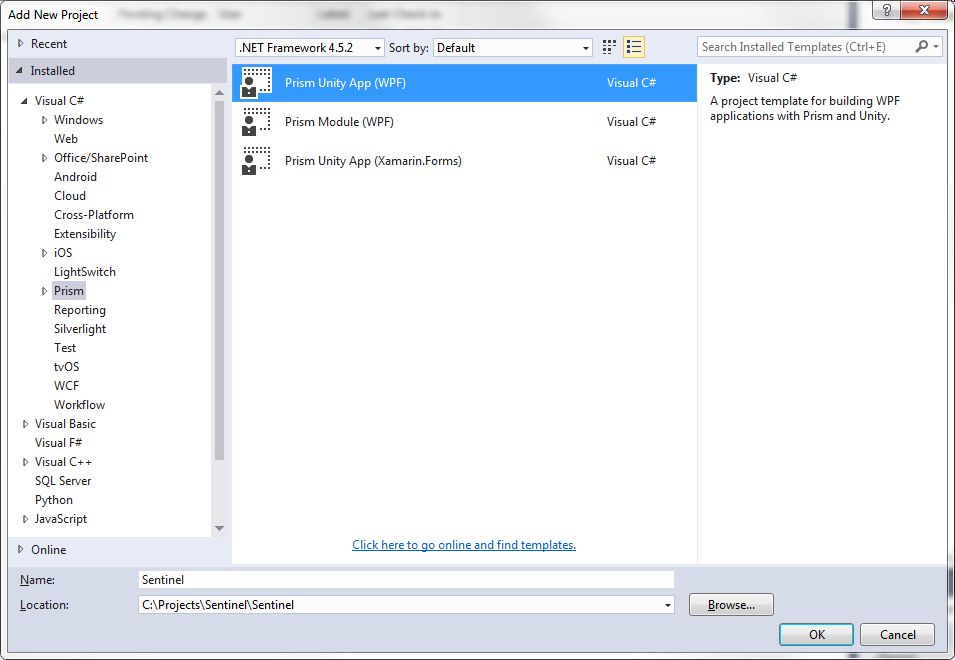
Fichier EntityType.cs.t4



Comment créer une solution

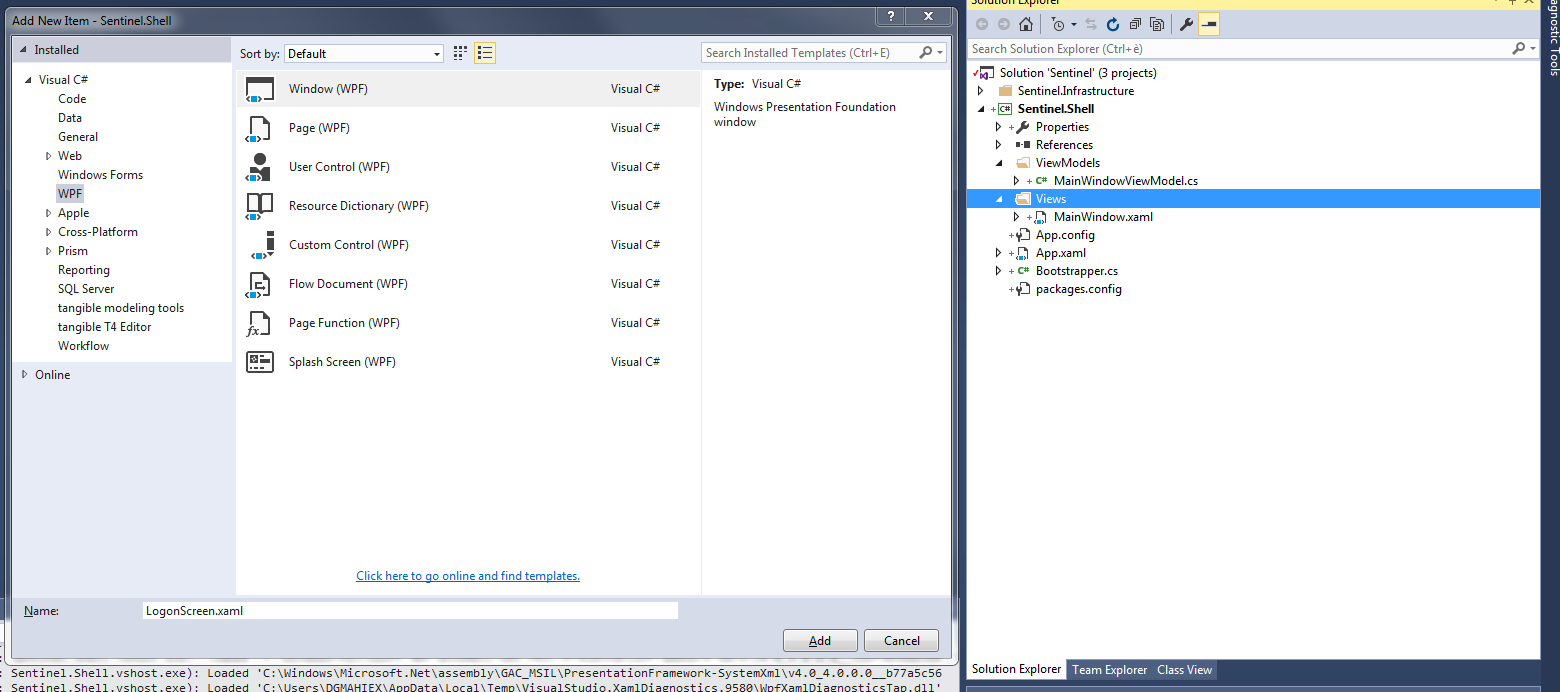


Creation d’un shell

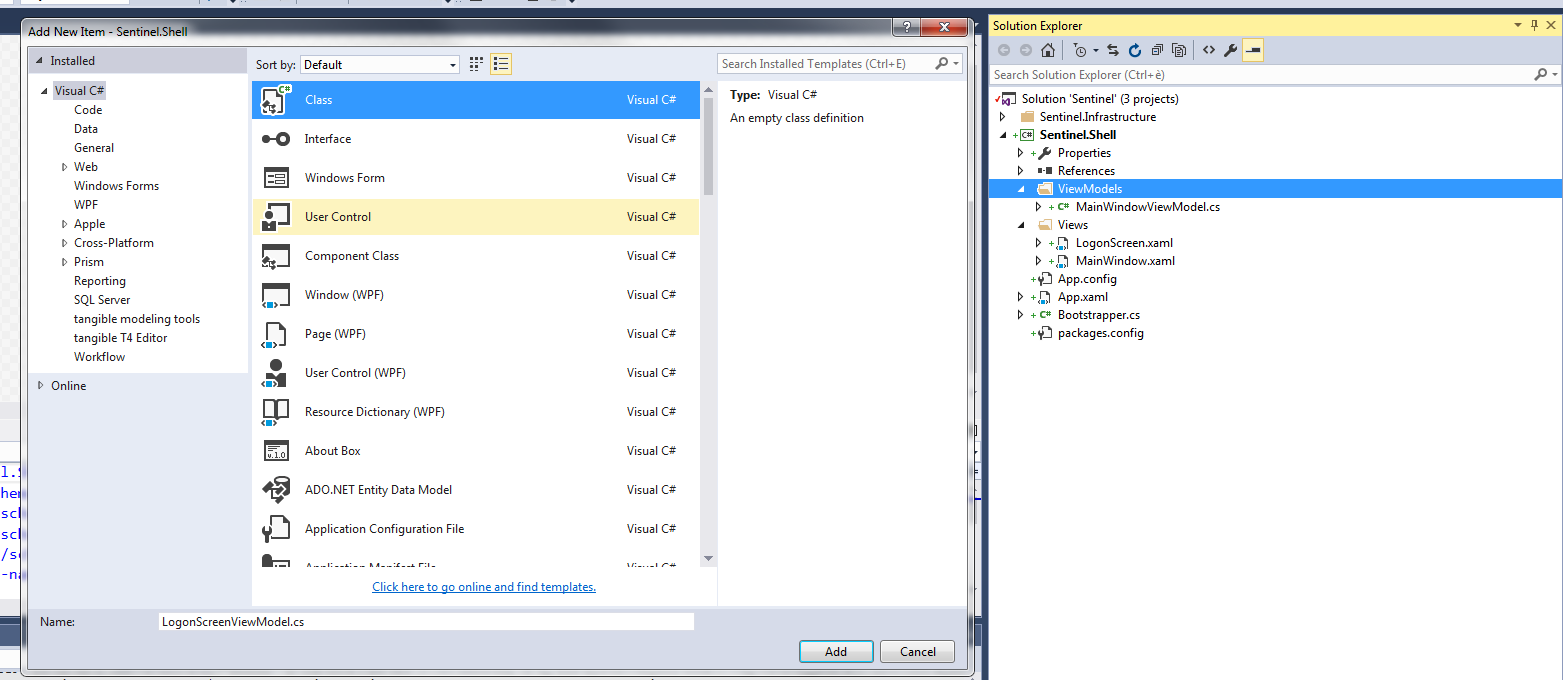


La création de l’application Shell entraîne la création de la fenêtre MainWindow.

Ajouter une vue



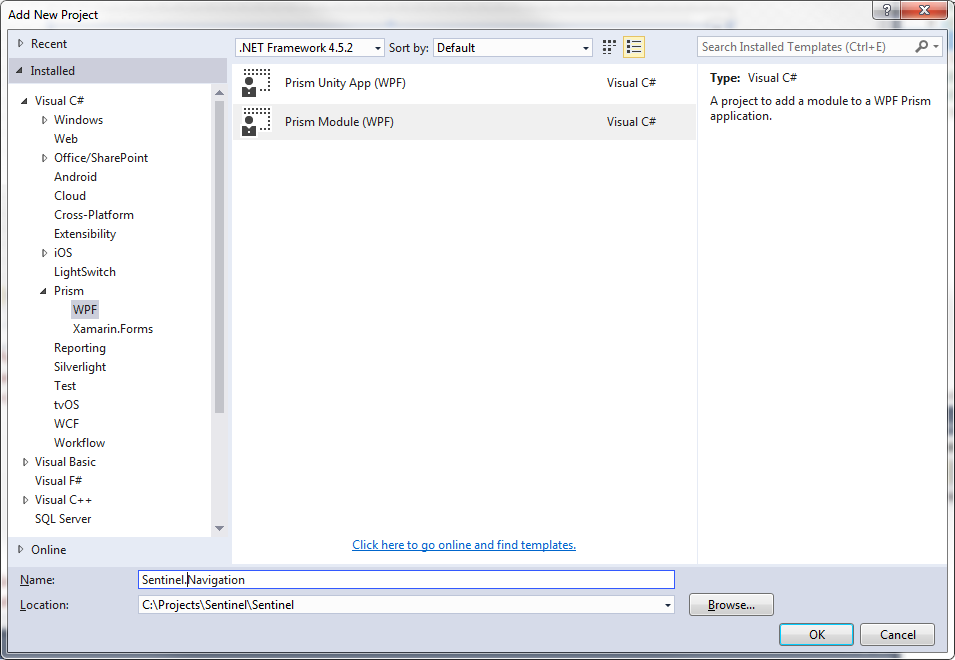
Ajouter une ViewModel



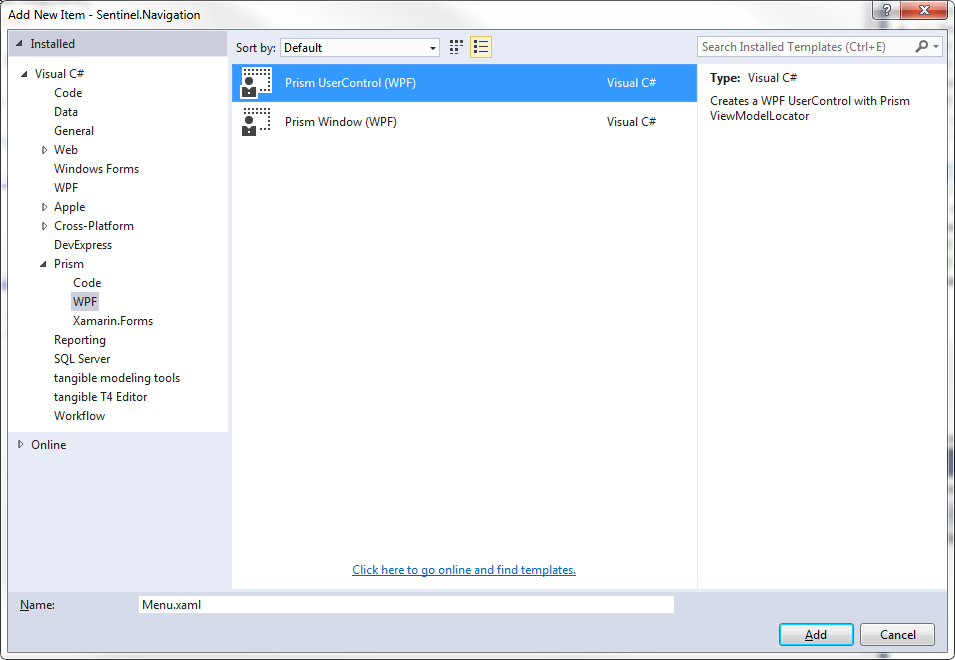
Prism

Comment ajouter un module

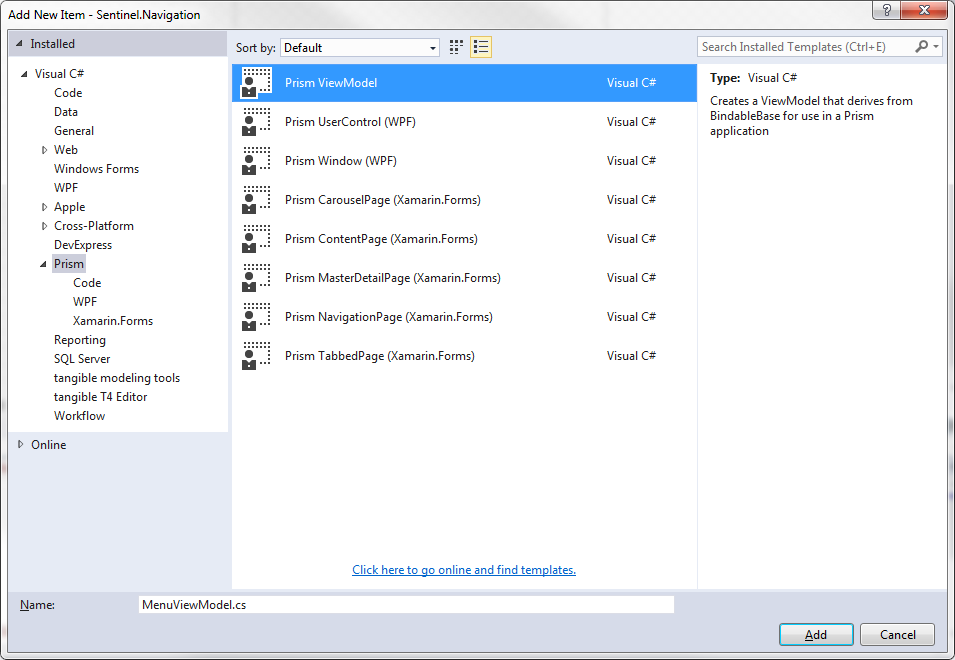
Add-> New Projet



## Ajout d’une vue dans le module



## Ajouter une ViewModel dans un module



# SOLUTION TECHNIQUE

## Choix effectués

Pour répondre aux différents besoins de l’application (desktop, modularité, design moderne, dynamisme des vues à afficher, navigation, communication entre les vues…), il a été décidé de s’orienter vers la technologie WPF en utilisant le framework PRISM et le pattern MVVM.

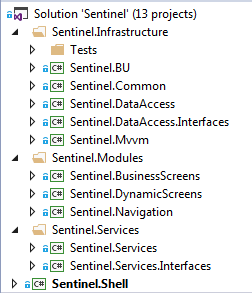
Version des outils

|  |  |
| --- | --- |
| Outils | Version |
| Visual Studio | 2015 |
| Team Foundation Services | 2010 |
| SQL server (standard edition) | 2012 |
| WPF | 4.5.2 |
|  |  |

|  |  |
| --- | --- |
| Librairies NuGet | Version |
| EntityFramework | 6.1.3 |
| EntityFramework.CodeTemplates.CSharp | 6.1.3 |
| Prism.Core | 6.2.0 |
| Prism.Unity | 6.2.0 |
| Prism.Wpf | 6.2.0 |
| CommonServiceLocator | 1.3.0 |
| Unity | 4.0.1 |
| T4 Editor | 2.3.0 |

## Organisation du code

Voici le code tel qu’organisé actuellement dans Visual Studio.



On peut distinguer 4 grands groupes :

* **Infrastructure**: Regroupe des fonctionnalités techniques (helpers, moteurs, accès données, …).
* **Modules** : Regroupement des vues et logiques de l’interface utilisateur.
* **Services** : Ensemble de fonctionnalités transverses à l’application.
* **Shell** : Interface utilisateur.

Liens entre ces groupes

MODULES

SHELL

INFRASTRUCTURE

SERVICES

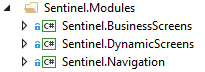
### SHELL



Point d’entrée de l’application, il initialise les différents composants nécessaires.

Le shell fournit également le socle de l’interface graphique (fenêtre principale MainWindow) et ses différentes régions.

### MODULES



**Business screens**

Traitement des écrans métier standards.

Ils seront dispatchés dans les 3 blocs de la fenêtre principale.

**DynamicScreens**

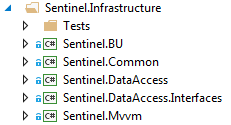
Traitement des écrans dynamiques (paramétrés spécifiquement pour le métier).

Ils seront dispatchés dans les 3 blocs de la fenêtre principale.

**Navigation**

Traitement des différents écrans de menu affiché dans la zone « Menu » de la fenêtre principale.

### INFRASTRUCTURE



**BU**

Logique métier et moteurs des différents processus.

**Common**

Regroupe un ensemble d’objets utiles pour la communication entre les modules et également toutes les énumérations, constantes, helpers et autres ressources statiques de l’application.

**DataAccess** / **DataAccess.Interfaces**

Centralise toutes les classes en lien avec la base de données (entités, opérations etc…)

### SERVICES



**Services / Services.Interfaces**

Ensemble de fonctionnalités transverses à toute l’application (logs, sécurité, audit, intermédiaire vers la base de données, etc…).

Ils sont injectés dans les modules via le principe des conteneurs « Unity ».

### Référencement entre projets



## PRISM

### Initialisation

* Bootstrapper

Dans le projet Shell nous créons la classe « BootStrapper.cs » qui va initialiser l’application (l’appel est effectué depuis le code behind de App.xaml).



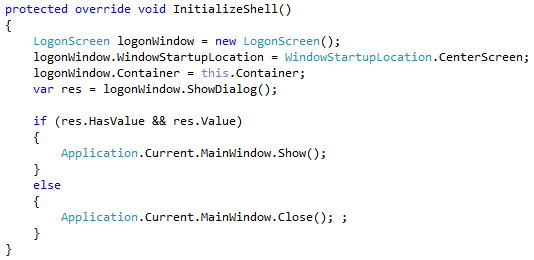
L’héritage de « UnityBootStrapper » va permettre l’utilisation de conteneurs « Unity » pour l’injection de dépendance.

* Récupération de l’instance de la fenêtre principale dans le conteneur.

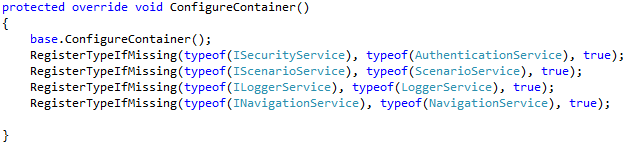


* Initialisation de la fenêtre principale.

On y intègre la notion d’écran d’authentification préalable à l’affichage.

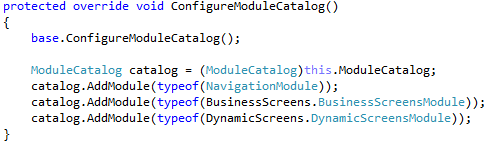


* Configuration du conteneur en procédant notamment au chargement de tous les services utiles à l’application via leur interface.



* Configuration du catalogue des modules nécessaires à l’application.

*Actuellement les modules sont défini dans la classe directement, il n’est pas exclu qu’ils le soient dans un fichier de configuration annexe pour plus de flexibilité.*



### Modularité

Chaque module de l’application est représenté par un projet contenant :

* Le point d’entrée et définition de ce module
* L’ensemble des vues inhérentes au module

Le point d’entrée est une classe centrale qui doit implémenter l’interface « PRISM » IModule.



C’est cette classe qui est ajoutée au catalogue des modules (cf. chapitre précédent).

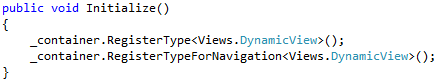
* Construction du module

On demande à « PRISM » de nous fournir le conteneur « Unity » transverse à l’application ainsi que le gestionnaire des régions (cf. points suivants).

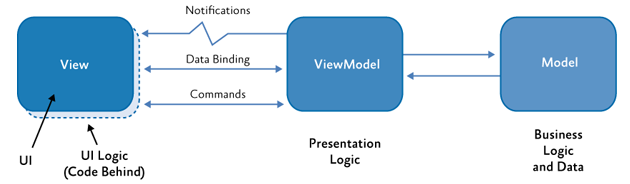


* Initialisation du module.

On charge dans le conteneur les vues utilisées et on les enregistre comme pouvant être affichées dans les différentes régions de l’application (principe de navigation).



### MVVM



* VIEW

Le système de vue est présent à la fois dans le Shell et dans les modules.

Le référencement du ViewModel par la Vue est automatiquement résolu grâce à la commande suivante dans le fichier XAML :



* VIEWMODEL

Tous les ViewModel liés à des modules héritent de la classe « SentinelViewModel » qui fournit des commandes et méthodes communes.





Les classes ViewModel échangent avec les Vues grâce à des propriétés et commandes qui sont « Bindés » sur les Vues.

Ces classes héritent de la classe abstraite « PRISM » BindableBase qui fournit une implémentation de l’interface InotifyPropertyChanged pour le binding des propriétés.



* MODEL

La partie Model est représentée par toutes les classes qui peuvent être impliquées dans le traitement et la validation des données. (Common, Services, Entités, DataAccess,…).

* BINDING

Comme énoncé ci-avant, il peut être effectué via des propriétés.

Toutes ces propriétés appellent l’implémentation de la méthode SetProperty (provenant de BindableBase)



Lorsque la propriété est du type collection, celle-ci doit être du type ObservableCollection<T>. Ce type a la particularité d’implémenter l’interface INotifyCollectionChanged qui permet de notifier la vue que le contenu de la liste a été modifié (ajout et suppression d’éléments, modification d’une valeur).



La source du binding est par défaut le datacontext de la Vue qui est dans notre cas le ViewModel.

On résout la propriété à binder dans la Vue via le mot clé « Path ».



Si l’on souhaite afficher le contenu d’une collection on peut utiliser des contrôles de type ItemsControl. Dans ce cas, le ViewModel ne sera plus la source du binding qu’il faut redéfinir via l’ItemsSource.



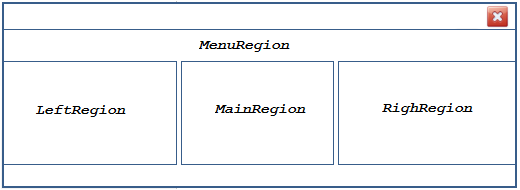
Dans un ItemsControl, il faut définir un modèle d’affichage des données. Pour chaque répétition du modèle, le bidning se fera sur l’élément courant de la collection.

S’il s’agit d’un objet, on précisera la propriété de cet objet via le mot clé « Path » comme décrit ci-avant.

### Interface utilisateur et navigation

* Définition des régions

L’interface est composée de 4 régions



Elles sont définies dans le code XAML de la fenêtre principale



* RegionManager

Afin de gérer les régions et la navigation avec PRISM, il faut utiliser une instance de la classe RegionManager fournie par PRISM.

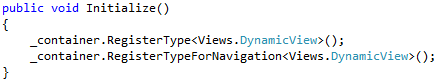
Elle peut être obtenue, dans le XAML, via la commande ci-avant énoncée et dans les modules via l’injection de dépendance.



* “View-based” navigation

La navigation à l’intérieur de l’application sera faite par le remplacement de Vues dans les différentes régions (principe de « view-based navigation » dans PRISM).

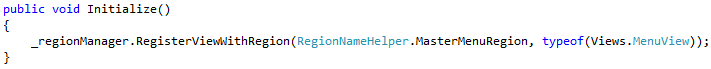
Dans chaque module, lorsque l’on charge les Vues dans le conteneur, on les enregistre également comme étant utilisées dans la navigation.



On utilise deux méthodes pour gérer l’affichage des vues dans les régions.

Le principe du « View discovery » qui va placer automatiquement la Vue dans une région lorsqu’elle est créée.

Cette définition est placée dans l’initialisation du module.



La méthode « View Injection » où l’on gère cet affichage dans le code au besoin.



Pour libérer une région de sa vue sans la remplacer, on peut utiliser



### Communication

* Event Aggregator

Ce service de PRISM permet la communication entre les différents modules de l’application, via un mécanisme de publication / souscription a des événements, sans que ceux-ci soient fortement liés.

Il peut être obtenu via l’injection de dépendance



Pour la création d’un événement spécifique à notre application, on crée une classe ayant pour classe de base « PubSubEvent<TPayLoad> » (où TPayLoad est le type du paramètre que l’on passera via l’événement).





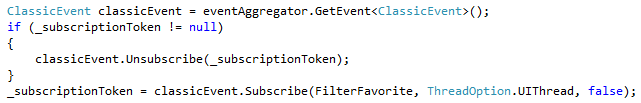
La publication de l’événement s’effectue en récupérant celui-ci et en le publiant via le service EventAggregator.





La souscription s’effectue via la méthode Subscribe de notre événement que l’on récupère au préalable.



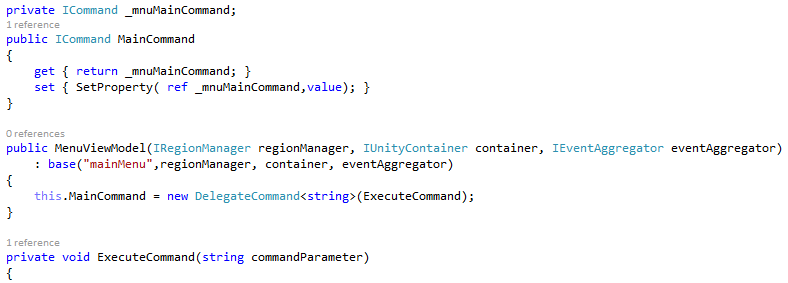


Dans notre code, on garde une trace de la souscription via son token. Par sécurité, annule potentielle souscription existante avant de créer la nouvelle.

L’utilisation de l’option ThreadOption.UIThread autorise le souscripteur à mettre à jour directement les éléments de l’interface utilisateur.

* Delegate command

On l’utilise pour la communication entre le ViewModel et la Vue

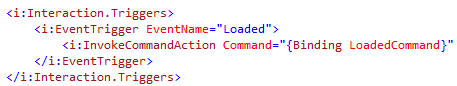


Dans la Vue

Si le control implémente l’interface ICommandSource

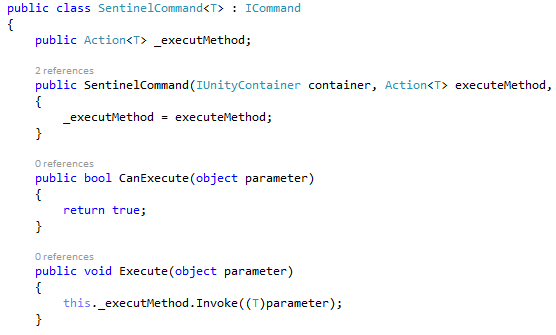


Sinon on utilise les « interaction triggers »



* Custom command



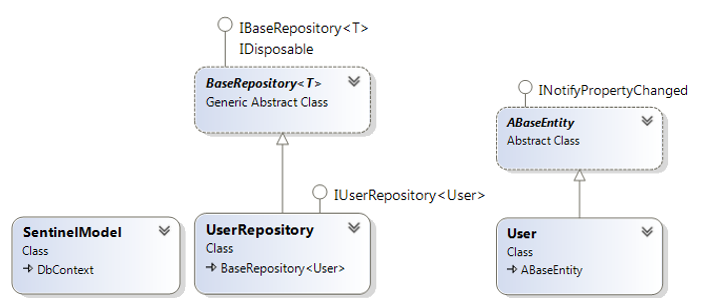


## Accès aux données

Pour la couche d’accès aux données, le choix s’est porté sur l’ORM Entity Framework.

L’approche DBFirst est utilisée et va générer automatiquement les composants (entités, repository, contexte) en fonction des tables de la base de données, via notamment des templates T4 .

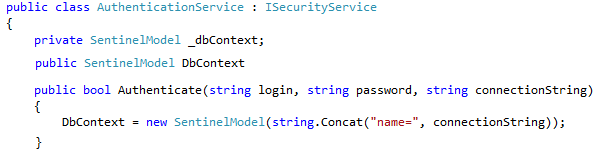
Le pattern repository est mis en place pour créer une couche d’abstraction entre les accès aux données et la logique business.



**SentinelModel** est la classe « contexte » qui expose les données de la DB sous forme de collection d’entités et définie les SP utilisées pour les opérations CRUD.



C’est le service d’authentification, premier instancié et injecté dans toute l’application, qui va fournir le contexte.



**Les entités** héritent toutes de la classe ABaseEntity qui implémente l’interface INotifyPropertyChanged afin de pouvoir utiliser le principe du binding sur leurs propriétés.

**Les classes Repository** sont créées pour chaque entité. Elles fournissent les opérations possibles sur les entités et utilisent le contexte pour interroger la base de données.



