Hervé PHILIPPE (alchiweb.fr)

Alchiweb.fr

Résumé

Document pour les développeurs, au sujet du template de génération de sites, AlchiFxCore

Date de création : 17/06/2019

AlchiFxCOre

Documentpour l’utilisation d’AlchiFxCore

Table des matières

[Préambule 2](#_Toc41401563)

[AlchiFxCore 2](#_Toc41401564)

[Présentation 2](#_Toc41401565)

[Utilisation 3](#_Toc41401566)

[Installation du modèle de projet 3](#_Toc41401567)

[Création du projet C# pour Visual Studio 3](#_Toc41401568)

[Modifications dans le projet 4](#_Toc41401569)

[A faire 6](#_Toc41401570)

[Tutoriel 7](#_Toc41401571)

[Création et configuration du projet 7](#_Toc41401572)

[Modèles et contexte de la base de données 7](#_Toc41401573)

[Génération des contrôleurs d’API 8](#_Toc41401574)

[Génération des contrôleurs MVC 8](#_Toc41401575)

[Remarques sur le code 9](#_Toc41401576)

[Choix des librairies JavaScript 9](#_Toc41401577)

[Choix des packages NuGet 10](#_Toc41401578)

[Fichier JavaScript « crud.js » 11](#_Toc41401579)

[Fichier « \_SpaEntitySignalScriptsPartial.cshtml » 12](#_Toc41401580)

[Projet Templating 13](#_Toc41401581)

# Préambule

Ce document a pour but de présenter « AlchiFxCore », outil permettant de générer automatiquement des sites Asp.Net Core avec Entity Framework Core.

# AlchiFxCore

## Présentation

**AlchiFxCore** est un template de projet C# permettant de créer des projets C# pour Visual Studio 2017 et+ :

Le site généré :

* Utilise principalement les technologies :
  + **Asp.Net Core** 3.1 (lui-même basé sur .Net Core 3.1) + **Entity Framework Core** 3.1
  + [Entity Signal](https://entitysignal.com/), pour mettre à jour automatiquement les données affichées dans des pages ouvertes
  + [BootStrap](https://getbootstrap.com/docs/4.3/getting-started/introduction/)
  + [jQuery](https://jquery.com/) + [jQueryUI](https://jqueryui.com/)
  + [jQuery Dialog](https://jqueryui.com/dialog/)  pour le multifenêtrage
  + [SlickGrid](http://slickgrid.net/) pour les tables
  + [Select2](https://select2.org/) (version compatible avec BootStrap 4), pour les listes déroulantes
* A un design basé sur [AdminLTE 3](https://adminlte.io/themes/dev/AdminLTE/index.html) (BootStrap 4).

Version en cours de finalisation : <https://github.com/ColorlibHQ/AdminLTE>

* Permet de gérer les utilisateurs (avec les pages de profil) :
  + Standard Asp.Net Users.
  + Intégration des pages avec le design d’AdminLTE
* Permet de gérer des tables de bases, avec du multifenêtrage ou en utilisant des pages dédiées.

A partir des modèles (et donc aussi du contexte de la base de données), seront générés :

* + Les contrôleurs web Api, gérant le CRUD (pour *create*, *read*, *update*, *delete*)
  + Les contrôleurs web gérant le CRUD
  + Les vues (pages Web) liées aux contrôleurs web et se servant des contrôleurs web Api.

Remarque : l’interface utilisateur s’inspire de celle utilisée dans le [projet opensource Serene](https://github.com/volkanceylan/Serene) (basé sur [Serenity](https://serenity.is/)). Par exemple : utilisation de jQuery Dialog, SlickGrid, Select2.

## Utilisation

### Installation du modèle de projet

En ligne de commande (cmd), lancer « **Install-AlchiFxCore** » dans le dossier d’AlchiFxCore :

>Install-AlchiFxCore

Remarques :

* A ne faire qu’une fois
* Pour une mise à jour :
  + Supprimer la version actuelle avec la ligne de commande :

>dotnet new -u Alchiweb.AlchiFxCore

* + Refaire l’installation
* Dans le dossier contenant « **Install-AlchiFxCore.cmd**», il y a aussi le fichier : « **Alchiweb.AlchiFxCore.2.2.1-servicing-t000.nupkg** »
* Dans le fichier « **-AlchiFxCore.cmd** », il y a juste la ligne :

dotnet new -i .\Alchiweb.AlchiFxCore.2.2.1-servicing-t000.nupkgc

### Création du projet C# pour Visual Studio

En ligne de commande (cmd), lancer « **Create-AlchiFxCore** » suivi du nom du projet à créer, dans le dossier d’AlchiFxCore.

Exemples :

>Create-AlchiFxCore Alchi.WebAppTest

>Create-AlchiFxCore ..\Alchi.WebAppTest

Remarques :

* Dans le dossier contenant « **Create-AlchiFxCore.cmd**», il y a aussi le fichier : « **Alchiweb.AlchiFxCore.2.2.2.2.1-servicing-t000.nupkg** »
* Dans le fichier « **Create-AlchiFxCore.cmd** », il y a juste la ligne :

dotnet new alchifxcore --auth Individual --use-local-db -o %1

### Modifications dans le projet

#### Copier/générer les modèles

* [optionnel] Pour générer le contexte et les modèles à partir de la base de données (Database first) :

Ex :

PM> Scaffold-DbContext "Data Source=sql2008;Initial Catalog=DataBaseName;User ID=sa;Password=\*\*\*;MultipleActiveResultSets=true" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -DataAnnotations -Force

* Pour les modèles (dans le dossier « Models ») :
  + Faire dériver les modèles de la classe « ModelBase ».
  + Conseil : créer une propriété « Id » en tant que Guid, pour TOUS les modèles.
  + Surcharger la méthode « ToString ». Par exemple :

public override string ToString() => $"{AdresseTexte}|{Complement}|{CodePostal}|{Ville}|{NomVille}|{Departement}";

Remarque : mettre des « *pipes* » pour afficher ces infos comme des colonnes d’un tableau, pour les listes déroulantes « Select » (gérées avec la lib JS « select2 »)

* Pour le contexte (dans le dossier « \Data » :
  + Utiliser la version générée.

Sinon, faire les changements suivants :

* + - AlchiDbContext qui hérite de EntitySignalIdentityDbContext<AlchiUser>

Remarque : AlchiUser doit hériter de IdentityUser.

* + - Constructeur :

public AlchiDbContext(DbContextOptions<AlchiDbContext> options, EntitySignalDataProcess entitySignalDataProcess) : base(options, entitySignalDataProcess)

* + - A la fin de la méthode « OnModelCreating », il faut :

base.OnModelCreating(modelBuilder);

#### Créer une nouvelle migration

Dans la ***Console du Gestionnaire de package NuGet***, taper :

Add-Migration PremiereMigrationAvecTablesDeBase

#### Mettre à jour la base de données

Par défaut, la base de données est en « **(localdb)**» (cf fichier « ***appsettings.json*** »).

Dans la ***Console du Gestionnaire de package NuGet***, taper :

Update-Database

#### Générer les controllers pour l’Api

Générer automatiquement le *controller* pour l’API web en fonction d’un modèle.

ATTENTION : enlever le « s » ajouté dans le nom du *controller*.

#### Générer les controllers et les vues

Générer automatiquement le *controller* en fonction d’un modèle.

ATTENTION :

* enlever le « s » ajouté dans le nom du *controller*.
* dans chaque *controller* généré, il faut déplacer la portion de code javascript (en haut du fichier du *controller*), dans le fichier « \wwwroot\js\DbTablesDef.js »

Une fois ces étapes faites, les vues https://**[NOM\_DU\_SITE]**/**[NOM\_DU\_MODELE]** sont générées et affichent la table de base du modèle.

## A faire

* Finir la mise à jour auto des fenêtres
  + Vérifier update pour des booléens (checkbox)
  + Vérifier update pour les enums (select)
  + À gérer : select (foreignkey + enums) dans la slickgrid ?????
* Finir Slickgrid:
  + filtrage, tri -> front-end + backend (Web Api via ajax)
  + modifications multiples, ajouter bouton « Sauver »
  + Afficher d’autres SlickGrids : sous-tables liées à un élément (ex : contacts…), affichables pour un élément (details, edit…), en étant filtrées suivant l’élément
* Logs des actions faites dans SQL.

Utilisation de LibLog (générique), Serilog ou NlLog… ?  
Stockage dans SQL ? En tant que donnée Json ?

Compatibilité RGPD ?

* Soft-delete (via Query filters dans EF Core) : lors de la suppression d’un élément, ne pas le supprimer réellement, mais le « flagguer » comme « IsDeleted » (ne plus le voir, sauf en mode « Log ». (par exemple : <https://www.ryansouthgate.com/2019/01/07/entity-framework-core-soft-delete/>)
* Bulk CRUD actions ([EFCore.BulkExtensions](https://github.com/borisdj/EFCore.BulkExtensions))  
  Par exemple, quand on modifie dans la SlickGrid plusieurs éléments
* [UI] Menu de gauche :
  + Enlever les liens vers les pages d’exemple d’AdminLTE
  + Ajouter automatiquement les pages du CRUD, pour chaque contrôleur MVC créé
* Gérer les rôles pour les autorisations.

Actuellement, dès qu’on est connecté, on a accès à toutes les pages. Il faudrait restreindre l’accès aux pages à certains rôles utilisateur (en utilisant les principes standard de Asp.Net « roles », « claims »…)

* Sécuriser les web Api avec JWT (tokens)
* Gérer toutes les librairies JS avec « npm » (package.json), pour pouvoir mettre à jour automatiquement le dossier « \wwwroot\plugins »
* Localisation : prévoir la gestion de plusieurs langues, grâce notamment à des ressources
* [optionnel] Création du fichier « crud.ts » (TypeScript) pour générer le fichier « crud.js »  
  De même pour « calendrier.js ».

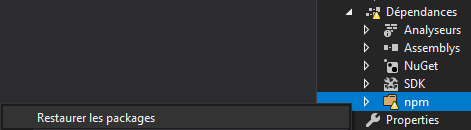
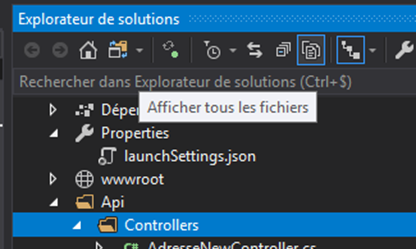
## Tutoriel

Suivent les explications détaillées, détaillant les explications du chapitre « [*Modifications dans le projet*](#_Modifications_dans_le) ».

### Création et configuration du projet

* **Créer le projet** via la commande « Create-AlchiFxCore »
* **Ouvrir le projet** dans Visual Studio 2019
* Restaurer des **packages npm**.

Dans « Dépendances -> npm », cliquer sur « Restaurer les packages » :



* Configuration de Visual Studio :  
  activer « **Afficher tous les fichiers** », pour pouvoir voir le dossier « Api\Controllers », qui servira à stocker les contrôleurs d’API (capture d’écran : avec les contrôleurs d’API déjà générés)

### Modèles et contexte de la base de données

* Copier :
  + Les 6 **modèles** (dans \Models)
  + Le **contexte** « AlchiDbContext.cs » (dans \Data)
* Créer une nouvelle migration :  
  Dans la ***Console du Gestionnaire de package NuGet***, taper :

Add-Migration PremiereMigrationAvecTablesDeBase

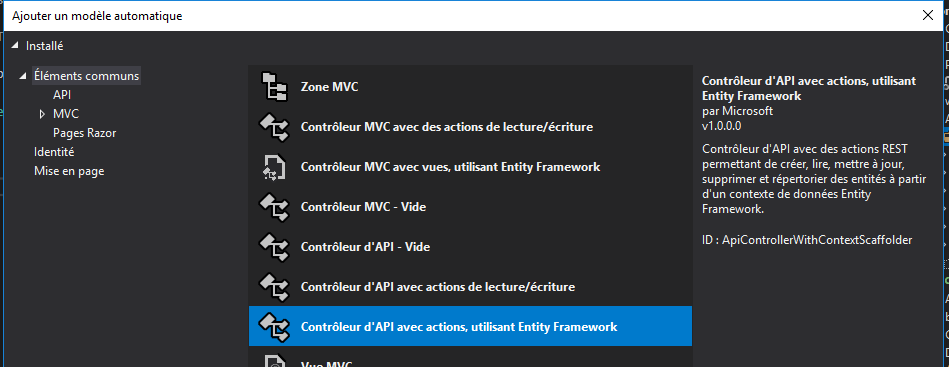
* Mettre à jour la base de données :  
  Dans la ***Console du Gestionnaire de package NuGet***, taper :

Update-Database

Remarque : par défaut, la base de données est en « **(localdb)**» (cf fichier « ***appsettings.json*** »).

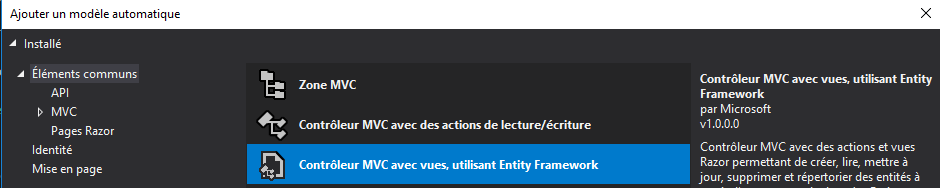
### Génération des contrôleurs d’API

Générer les **contrôleurs d’API** (dans VS), pour chaque modèle

* Sur « Api\Controllers », cliquer sur « Ajouter -> Nouvel élément généré automatiquement… »
* Choisir « Contrôleur d’API avec actions, utilisant Entity Framework »
* Cliquer sur « Ajouter »
* Choisir la classe de modèle ET enlever le « s » pour le nom du contrôleur
* Cliquer sur « Ajouter »

### Génération des contrôleurs MVC

Générer les **contrôleurs MVC** (dans VS), pour chaque modèle :

* Sur « Controllers », cliquer sur « Ajouter -> Nouvel élément généré automatiquement… »
* Choisir « Contrôleur MVC avec vues, utilisant Entity Framework »
* Cliquer sur « Ajouter »
* Choisir la classe de modèle ET enlever le « s » pour le nom du contrôleur
* Cliquer sur « Ajouter »
* Une fois le contrôleur généré, **couper/coller** la portion de code JavaScript du fichier « .cs » du contrôleur généré vers le fichier « DbTablesDef.js ».

Remarque :

* le fichier « DbTablesDef.js » sert à la création des variables « optionsShowCrud\_[NomDuModèle] », servant pour les fonctions JavaScript.

Par exemple, pour définir :

* + Les colonnes affichées dans la SlickGrid
  + Le nom du modèle associé
  + Les libellés pour le CRUD
  + …

## Remarques sur le code

### Choix des librairies JavaScript

* Pour les tableaux :
  + [jQuery DataTables](https://datatables.net/) a été utilisé au départ car :
    - intégré dans AdminLTE
    - très utilisé
    - mais… partie payante (fonction « grid » modifiable)

*Remarque : un prototype avait été fait*

* *utilisation de* [*JqueryDataTablesServerSide*](https://github.com/fingers10/JqueryDataTablesServerSide)*, pour Web Api (filtres, tris…)*
* *utilisation de* [*AutoMapper*](https://automapper.readthedocs.io/en/latest/Getting-started.html) *pour mapper automatiquement les* ***modèles*** *et les* ***objets Json*** *servants à jQuery DataTables*
  + [SlickGrid](http://slickgrid.net/) a été utilisé car :
    - totalement opensource,
    - existe depuis longtemps (initié par [mleibman](https://github.com/mleibman/SlickGrid/wiki), mais repris par [6pac](https://github.com/6pac/SlickGrid/wiki))
    - exemple d’utilisation dans Serene (extension dans Visual Studio)

*Remarque : il serait intéressant de faire comme pour le prototype avec jQuery DataTables, pour gérer les filtres, tris, modifications…*

* Pour les calendriers :
  + [FullCalendar](https://fullcalendar.io/) :
    - intégré dans AdminLTE
    - très utilisé
    - mais… « FullCalendar Scheduler » est payant

*Remarque : pour certains projets, l’affichage en «*[*Resource Timeline*](https://fullcalendar.io/#demos)*» est utile mais nécessite d’acheter « FullCalendar Scheduler ».*

* + Autres possibilités : après recherches, il est assez compliqué de tomber sur une librairie complètement opensource proposant l’affichage avec les heures en colonnes.
* Pour les listes déroulantes :
  + [Select2](https://select2.org/) :
    - totalement opensource,
    - exemple d’utilisation dans Serene (extension dans Visual Studio)
    - maintenant compatible avec BootStrap 4

### Choix des packages NuGet

* [Enums.NET](https://github.com/TylerBrinkley/Enums.NET) n’est actuellement pas utilisé, mais a de nombreux avantages pour manipuler les enums
* [Humanizer](https://github.com/Humanizr/Humanizer) est généralement très utile, pour diverses raisons.

Il est utilisé pour le template de génération des contrôleurs MVC, mais n’est actuellement pas utilisé pour les projets générés.

* [EntitySignal.Server](https://github.com/dustout/entitysignal)

Un *fork* a été fait, pour rendre cette version compatible avec Asp.Net Core 3.1 et Entity Framework Core 3.1.

* La partie « **Client** » de **EntitySignal** a été intégrée au code de l’application générée :
  + Ajout de la méthode C# « **SubscribeToAll** » dans les contrôleurs MVC générés

*Remarque : il faudrait faire d’autres méthodes, pour faire une inscription sur une partie des éléments (par exemple filtrés) d’une liste.  
Ça pourrait être utile dans le cas de l’affichage (ou d’édition) d’une page d’un élément ayant une jointure vers une autre table.*

* + Fichier TypeScript de la classe « **EntitySignal** » (entitySignal.ts)
  + Fichier « \Views\Shared\**\_SpaEntitySignalScriptsPartial.cshtml** » où sont définies :
    - La **variable globale « clientEntitySignal** » objet instanciant la classe « EntitySignal » (entitySignal.ts)
    - La **fonction globale « pageCrudItem »,** appelée dans la méthode « processSync » de la classe « EntitySignal » (entitySignal.ts).

function pageCrudItem(entityTypeName, action, item, idColumnName)

* + Fichier « **crud.js** » : utilisation en JS de la variable « **clientEntitySignal** », objet de classe EntitySignal

### Fichier JavaScript « crud.js »

Le fichier « \wwwroot\js\crud.js » sert à la gestion dus CRUD côté client, avec des fonctions JavaScript.

#### Fonction InitCrud

Cette fonction permet de créer (et de gérer) un tableau SlickGrid, lié à un modèle de données.

**function InitCrud****(entityForOptions, idSource, idButtonsCrud = null)**

***Remarques****:*

* *On peut créer différents tableaux dans la même page.*
* *Si on a besoin de créer des tableaux dans les pages « Create », « Delete », « Details » ou « Edit », il ne faut pas oublier de décommenter les lignes :*

*//await Html.RenderPartialAsync("\_SlickGridStylePartial"); // pour la grille*

*//await Html.RenderPartialAsync("\_SlickGridScriptsPartial"); // pour la grille*

**Paramètres :**

* « **entityForOptions** » : suffixe permettant d’en déduire l’utilisation de la variable globale « optionsShowCrud\_[entityForOptions] », définie dans le fichier « \wwwroot\js\DbTablesDef.js »

*Remarque : « entityForOptions » peut avoir un autre nom qu’un modèle de données.  
Ainsi, si on a généré un contrôleur basé sur un modèle de données et qu’on génère un autre contrôleur basé sur ce même modèle (avec un autre nom -> nom qui ne correspond donc pas au nom du modèle de données), on pourra avoir une table SlickGrid différente (par exemple avec un filtre différent, ou des colonnes différentes).*

* « **idSource** » : id de l’élément Html où sera généré la table SlickGrid.

Par exemple, dans les fichiers « Index.cshtml », c’est un « div » (à la fin du fichier).

* [optionnel] « **idButtonsCrud** » : id de l’élément Html où sera généré le bouton « Créer ».  
  Par exemple, dans les fichiers « Index.cshtml », c’est un « div » (dans les boutons de la barre d’outils).

#### Fonction ShowCrud

Cette fonction sert à afficher une page du CRUD, en mode « multifenêtrage » :

* Soit à partir du menu contextuel d’une SlickGrid
* Soit à partir de boutons-liens ayant la classe CSS « crud\_indialog »

function ShowCrud(e)

L’idée est de récupérer les informations suivantes, qui sont utiles pour cette fonction :

* Variable globale des options de ShowCrud (« optionsShowCrud\_... »)
* Action pour le CRUD (Edit, Details, Create, Delete)
* Id de l’élément (si l’action n’est pas Create)

#### Fonction GetForeignValue

Cette fonction sert à récupérer, si possible, les informations d’un élément venant d’une clé secondaire.  
Pour ce faire, on va recherche dans la table des « subscriptions » de l’objet « clientEntitySignal », pour voir si l’élément est présent.  
En effet, lors de chaque souscription (via EntitySignal), la liste des éléments (filtré ou non, suivant le type de souscription) est sauvegardé et mis à jour automatiquement.

#### Variable globale optionsSelect2

La variable globale « optionsSelect2 » (définie au début du fichier crud.js) sert à définir le comportement par défaut des listes déroulantes basées sur « select2 ».

Sont notamment définies les options :

* « templateSelection », pour l’affichage de la sélection courante
* « templateResult », pour l’affichage de éléments de la liste déroulante.  
  A noter : le caractère ‘|’ sert à délimiter les informations, pour les afficher dans des colonnes dans la liste déroulante.

L’information affichée est la propriété « DisplayValue », qui correspond à la surcharge du « ToString » dans les modèles de données -> il est conseillé d’ajouter les différentes valeurs utiles pour définir un élément, ces valeurs étant séparées par des ‘|’.

#### Variables « gridTab\_[Entity] » et « gridTabDataView\_[Entity] »

Ces variables sont générées dynamiquement. Elles sont utiles pour gérer la mise à jour automatique des éléments dans la page (via EntitySignal).

* « gridTab\_[Entity] » : stocker toutes les « SlickGrid » de la page
* « gridTabDataView\_[Entity] » : stocker toutes les « Slick.Data.DataView » de la page

### Fichier « \_SpaEntitySignalScriptsPartial.cshtml »

Ce fichier sert à mettre en place les choses indispensables pour l’utilisation de EntitySignal sur une page.

Sont définis :

* La variable globale « clientEntitySignal », objet instanciant la classe « EntitySignal ».
* La fonction globale « pageCrudItem », qui définit le comportement lors de l’arrivée d’un événement EntitySignal

## Projet Templating

Le projet « Templating » sert à la création d’un package NuGet qui va servir à générer des projets, basés sur un modèle de projet.

La version officielle de la [solution Templating d’Asp.Net Core 2.2](https://github.com/aspnet/Templating/tree/release/2.2) a servi de base pour ce projet.

Le modèle de projet « [StarterWeb-CSharp](https://github.com/aspnet/Templating/tree/release/2.2/src/Microsoft.DotNet.Web.ProjectTemplates/content/StarterWeb-CSharp) » a été modifié et renommé en « AlchiFxCore-CSharp ».

Ainsi, c’est dans le dossier « \src\Alchi.AlchiFxCore\content\AlchiFxCore-CSharp » que l’on trouve tous les fichiers nécessaires au modèle de projet.

Remarques :

* Le *namespace* de ce modèle de projet est « **Company.WebApplication1** ». A la création d’un nouveau projet, on spécifiera le nom du projet (qui sera en fait le namespace du projet), et tous les « **Company.WebApplication1** » seront remplacés par le nouveau namespace.
* Le dossier « **Templates** » dans un projet, permet de surcharger les templates de création des pages automatiques (par exemple, pour le MVC).
  + Il faut exclure du projet le dossier « Templates ». Par exemple (dans le fichier .csproj) :

<ItemGroup>

<Compile Remove="libs\\*\*" />

<Content Remove="libs\\*\*" />

<Content Remove="Templates\\*\*" />

<EmbeddedResource Remove="libs\\*\*" />

<EmbeddedResource Remove="Templates\\*\*" />

<None Remove="libs\\*\*" />

<None Remove="Templates\\*\*" />

</ItemGroup>

* + Pour revenir à un comportement standard de génération de pages MVC dans Visual Studio, il suffit de supprimer le dossier « Templates »
  + La syntaxe des fichiers .cshtml est particulière… et l’Intellisense ne fonctionne pas.
* Attention lors d’un « copier-coller » : s’il y a un changement automatique, par exemple pour les indentations et les sauts de ligne, il risque fort d’y avoir un soucis ! Faire « ctrl-z » après le « coller » permet de rétablir les choses.
  + Un package NuGet est utilisé cette génération à partir des fichiers du dossier « Templates ».

C’est peut-être « microsoft.visualstudio.web.codegenerators.mvc » (dans « C:\Users\[Nom\_Utilisateur]\.nuget\packages »).

A partir du projet « [Asp.Net / Scaffolding](https://github.com/aspnet/Scaffolding) » on peut créer une autre version du générateur.

En modifiant le projet « VS.Web.CG.Mvc.csproj » (dans « \src\VS.Web.CG.Mvc »), on peut créer une autre DLL pour par exemple prendre en compte un autre fichier à générer… mais ça nécessitait de changer le comportement général de Visual Studio pour la génération des MVC, donc cette option a été abandonnée.  
C’est pour cette raison que une partie de JavaScript a été générée dans le fichier du contrôleur MVC.