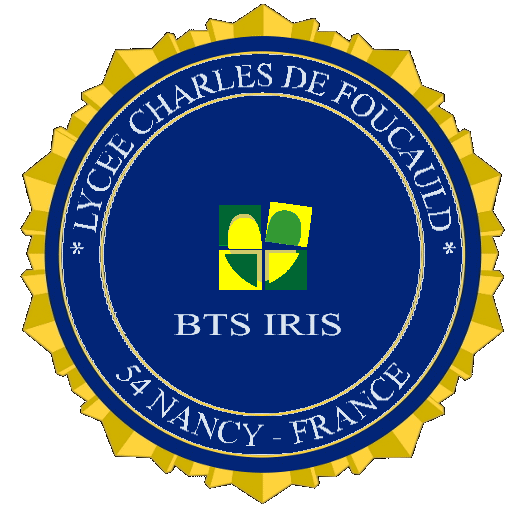
|  |
| --- |
| **NorthWind 54** |
| Dossier Technique |
| Charles de Foucauld - NANCY |
| Académie de Nancy-Metz Version : V 1 |
| Réf : -00- cdfDT\_GTTW2014.docx |
| **José ALVAREZ** |
| **12/02/2014** |
|  |

Projet BTS IRIS

**Session : xxxx**

Etudiants :

|  |
| --- |
| ***Etudiant N°1 =*** |
| Etudiant N°2 = |
| Etudiant N°3 = |
| ~~Etudiant N°4 =~~ |

# Historique des modifications du document

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Auteur** |
| 2012-01-06 | 0.0 | * Modèle de document. | J. ALVAREZ |
| 2014-02-20 | 1.0 | * Version 17 = V1.2.5.5277 [1.0.Alpha] * Version initiale | J.ALVAREZ |
| 2014-02-22 | 1.1 | * Version 18 = V1.2.5.5302 [1.1.Alpha] * Mise à jour de Category (View & Controller) * Ajout de ValidateAntiForgeryToken dans sécurité | J.ALVAREZ |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

SOMMAIRE

[1. Historique des modifications du document 4](#_Toc380856945)

[SOMMAIRE 5](file:///C:\Users\jow\Documents\Visual%20Studio%202013\Projects\NorthWind54\Documentation\-00-%20cdfDT_GTTW2014.docx#_Toc380856946)

[2. Cahier des charges 8](#_Toc380856947)

[2.1 Cahier de charges initial 8](#_Toc380856948)

[2.1.1 Besoins fonctionnels. 8](#_Toc380856949)

[2.1.2 Besoins non fonctionnels. 8](#_Toc380856950)

[2.2 Interviews 9](#_Toc380856951)

[2.3 Réunion du : 9](#_Toc380856952)

[2.4 Cahier des charges final 10](#_Toc380856953)

[2.4.1 Document contractuel 10](#_Toc380856954)

[3. Analyse 11](#_Toc380856955)

[3.1 Définition des acteurs 11](#_Toc380856956)

[3.1.1 Acteurs du produit 11](#_Toc380856957)

[3.1.2 Acteurs Scrum. 11](#_Toc380856958)

[3.2 Analyse des besoins (Product Backlog) 12](#_Toc380856959)

[3.3 Diagramme de contexte 13](#_Toc380856960)

[3.4 Description des cas d’utilisation 14](#_Toc380856961)

[3.4.1 Diagrammes des cas d’utilisation. 14](#_Toc380856962)

[**Cas d’utilisation : « Nom du cas d’utilisation »** *Utiliser une tournure à l’infinitif (ex : Réceptionner un colis).* 14](#_Toc380856963)

[3.5 Diagrammes de séquences ou/et de collaboration 17](#_Toc380856964)

[4. PROTOTYPAGE DES I.H.M. 18](#_Toc380856965)

[5. Schéma des bases de données. 19](#_Toc380856966)

[5.1 ASP.NET Identity 19](#_Toc380856967)

[5.2 NorthWind 20](#_Toc380856968)

[6. Architecture logicielle 21](#_Toc380856969)

[7. Architecture matérielle 23](#_Toc380856970)

[8. Planification des tâches 24](#_Toc380856971)

[8.1 Tâches globales 24](#_Toc380856972)

[8.2 Détails des tâches 24](#_Toc380856973)

[9. Pré développement 25](#_Toc380856974)

[9.1 Mise en place de la solution sur le poste de développement 25](#_Toc380856975)

[9.2 Contrôle de code source 26](#_Toc380856976)

[9.2.1 Création des comptes sur Codeplex et Github 26](#_Toc380856977)

[9.2.2 Projet sur Codeplex et Visual Studio 26](#_Toc380856978)

[10. Conception 27](#_Toc380856979)

[10.1 Home controller class diagram 27](#_Toc380856980)

[10.2 Account Class Diagram. 28](#_Toc380856981)

[10.3 Category/Product Class Diagram 29](#_Toc380856982)

[10.4 NorthWind class diagram 30](#_Toc380856983)

[10.5 HomeController class diagram 31](#_Toc380856984)

[10.6 Codage 32](#_Toc380856985)

[10.6.1 Internationalisation 32](#_Toc380856986)

[10.6.1.1 Mécanismes 32](#_Toc380856987)

[10.6.1.2 Architecture 32](#_Toc380856988)

[10.6.1.3 Modification des messages 33](#_Toc380856989)

[10.6.1.4 Localisation des messages d'erreur de validation et DisplayAttribute 33](#_Toc380856990)

[10.6.2 Trace et logs 34](#_Toc380856991)

[10.6.2.1 Centralisation de l’écriture des traces et des logs 34](#_Toc380856992)

[10.6.2.2 Filtre 35](#_Toc380856993)

[10.6.2.3 Trace 36](#_Toc380856994)

[10.6.2.4 System.Diagnostics.Trace 37](#_Toc380856995)

[10.6.2.5 Elmah 37](#_Toc380856996)

[10.6.2.6 Log4Net 37](#_Toc380856997)

[10.6.3 Gestion des performances avec Glimpse 37](#_Toc380856998)

[10.6.4 Gestion des exceptions 38](#_Toc380856999)

[10.6.4.1 Elément « customErrors » de Web.Config 38](#_Toc380857000)

[10.6.4.2 filters.Add(new HandleErrorAttribute()); 38](#_Toc380857001)

[10.6.4.3 Cas particulier des exceptions gérées 39](#_Toc380857002)

[10.6.4.4 « Controller » MyError 40](#_Toc380857003)

[10.6.4.5 « View » correspondant aux actions 41](#_Toc380857004)

[10.6.4.6 Pour les tests : 41](#_Toc380857005)

[10.6.5 « Category » et « Product » 42](#_Toc380857006)

[10.6.5.1 « Controller » 44](#_Toc380857007)

[10.6.5.2 « Model » 45](#_Toc380857008)

[10.6.5.3 « View » 46](#_Toc380857009)

[10.6.5.4 « ViewModel » 47](#_Toc380857010)

[10.6.5.5 « TrunckedCategories View » 48](#_Toc380857011)

[10.6.6 ASP.NET Identity & Extension 49](#_Toc380857012)

[10.6.6.1 Microsoft ASP.NET Identity 49](#_Toc380857013)

[10.6.6.2 Extension des comptes Identity et implémentation des rôles 52](#_Toc380857014)

[***10.6.6.2.1*** ConnectionString 52](#_Toc380857015)

[10.6.6.2.2 Configuration de la « Migration » Entity FrameWork 52](#_Toc380857016)

[10.6.6.2.3 Suppression des fichiers et du code non utilisés par l’application 53](#_Toc380857017)

[10.6.6.2.4 Extension des « Models » et des « View Models » 54](#_Toc380857018)

[10.6.6.2.5 Extension du « Controller » Account. 58](#_Toc380857019)

[10.6.6.2.6 Basiques « Views » pour le management d’« Identity » basé sur les rôles. 61](#_Toc380857020)

[10.6.6.2.7 Paramétrage et exécution des Migrations Entity Framework 63](#_Toc380857021)

[10.6.6.3 Ajout d’une photo dans le profil utilisateur 65](#_Toc380857022)

[10.6.6.3.1 Modification de la classe : « ApplicationUser » 65](#_Toc380857023)

[10.6.6.3.2 Modification de la classe : « RegisterViewModel » 65](#_Toc380857024)

[**10.6.6.3.3** Ajout d’une méthode static /Helpers/Utils/**SavePhotoFileToDisk** 66](#_Toc380857025)

[10.6.6.3.4 Modification de « Register » de : « AccountController » 67](#_Toc380857026)

[10.6.6.3.5 Modification de : « /Views/Acount/Register.cshtml » 67](#_Toc380857027)

[10.6.6.3.6 Modification de : « /Views/Acount/Index.cshtml » 67](#_Toc380857028)

[10.6.6.3.7 Modification de la classe : « EditUserViewModel ». 68](#_Toc380857029)

[10.6.6.3.8 Modification de la méthode « Edit » de : « AccountController » 68](#_Toc380857030)

[10.6.6.3.9 Modification de « DeleteConfirmed» de : « AccountController » 68](#_Toc380857031)

[10.6.6.3.10 Modification de : « /Views/Account/Edit.cshtml » 69](#_Toc380857032)

[10.6.6.3.11 Modification de : « /Views/Account/Delete.cshtml » 69](#_Toc380857033)

[10.6.6.3.12 Mise à jour de la base de données : 70](#_Toc380857034)

[10.6.6.3.13 Ajout modification du profil par un utilisateur 71](#_Toc380857035)

[10.6.6.3.13.1 Modification des « Views » 71](#_Toc380857036)

[10.6.6.3.13.2 Modification de AccountController.cs 71](#_Toc380857037)

[10.6.6.3.14 Ajout DataAnnotation : [ValidateFile] 72](#_Toc380857038)

[10.6.7 Design & Razor & BootStrap & Rotativa 74](#_Toc380857039)

[10.6.7.1 Viewport 74](#_Toc380857040)

[10.6.7.2 Table Category 74](#_Toc380857041)

[10.6.7.3 PDF : Rotativa 77](#_Toc380857042)

[10.6.7.4 Googlemaps dans contact.cshtml 78](#_Toc380857043)

[10.6.7.5 Razor HtmlHelper déclaratif ou méthode d’extension de HtmlHelper 79](#_Toc380857044)

[10.6.8 Area « Administration » 81](#_Toc380857045)

[10.6.8.1 Areas/Administration/Controllers/Home 82](#_Toc380857046)

[10.6.8.2 Areas/Administration/Views/Index.cshtml 82](#_Toc380857047)

[10.6.8.3 Areas/Administration/Views/Shared/\_Layout 83](#_Toc380857048)

[10.6.9 WebApi 84](#_Toc380857049)

[10.6.9.1 App\_Start/ WebApiConfig.cs 84](#_Toc380857050)

[10.6.9.2 Les « Controllers » implémentés sont dans le répertoire « WebApi ». 85](#_Toc380857051)

[10.6.9.3 Clients WebApi 85](#_Toc380857052)

[10.6.9.3.1 Clients web html 85](#_Toc380857053)

[10.6.9.3.2 Clients web MVC avec JQuery 86](#_Toc380857054)

[10.6.9.3.3 Clients web MVC avec accès à WebApi depuis le controller » 87](#_Toc380857055)

[10.6.9.3.4 Liens dans \_Layout 88](#_Toc380857056)

[10.6.9.3.5 Client Console 89](#_Toc380857057)

[10.6.9.3.6 Client WebForm 90](#_Toc380857058)

[10.6.10 Sécurité 91](#_Toc380857059)

[11. Déploiement 91](#_Toc380857060)

[11.1 Espace de nom. 91](#_Toc380857061)

[11.2 Archivage. 91](#_Toc380857062)

[11.3 Profils de publication et Web.Config associés 91](#_Toc380857063)

[11.4 Intégration de la base Identity à la base NorthWind 92](#_Toc380857064)

[11.5 « Migrations » Staging/Production 92](#_Toc380857065)

[11.5.1 Script de génération de la base de données 92](#_Toc380857066)

[11.5.2 Migration « Initialisation » sur le serveur « Staging » [IKOULA] 95](#_Toc380857067)

[11.5.3 Script de génération de mise à jour pour l’ajout de photo 96](#_Toc380857068)

[11.5.4 Migration « Photo » sur le serveur « Staging » [IKOULA] 97](#_Toc380857069)

[12. Recommandations 98](#_Toc380857070)

[**12.1** **Instructions relatives aux noms.** 98](#_Toc380857071)

[**12.2** **Conventions de dénomination du projet.** 98](#_Toc380857072)

[**12.2.1** **Sous-systèmes.** 98](#_Toc380857073)

[12.3 Ressources et Settings 98](#_Toc380857074)

[12.4 Rédaction des dossiers 98](#_Toc380857075)

[13. Journal de bord 99](#_Toc380857076)

# Cahier des charges

## Cahier de charges initial

### Besoins fonctionnels.

### Besoins non fonctionnels.

## Interviews

## Réunion du :

## Cahier des charges final

### Document contractuel

|  |  |  |  |
| --- | --- | --- | --- |
| Fait à Nancy le | | | |
| Prénom Nom | Société | Fonction | Signature |
|  |  |  |  |
| M. José ALVAREZ | CDF54 | Responsable du projet |  |
|  |  | Développeur |  |
|  |  | Développeur |  |
|  |  | Développeur |  |

# Analyse

## Définition des acteurs

### Acteurs du produit

|  |  |
| --- | --- |
| Intitulé | Commentaire |
| Invité | Consultatif |
| Administrateur |  |
| Base de données Authentification | NorthWind |
| Base de données NorthWind | NorthWind |

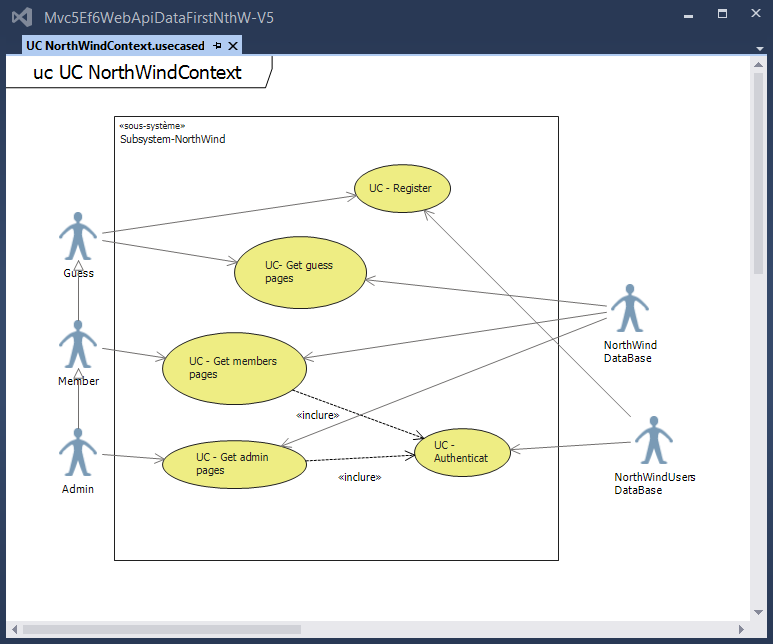
### Acteurs Scrum.

|  |  |  |  |
| --- | --- | --- | --- |
| Rôle | NOM Prénom | Organisation | Sous-Système |
| Product Owner |  |  |  |
| Scrum Master | *M. José ALVAREZ* | CDF54 |  |
| Développeur |  |  |  |
| Développeur |  |  |  |
| Développeur |  |  |  |

## Analyse des besoins (Product Backlog)

|  |  |  |  |
| --- | --- | --- | --- |
| Users & Technicals Stories (Récits utilisateurs) | | | |
| **Sprint** | **En tant que…** | **Je désire…** | **Afin de…** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Diagramme de contexte



## Description des cas d’utilisation

[Les classes d’analyse sont découvertes scénarios après scénarios, au moyen d’objets qui, en collaborant, réalisent des cas d’utilisation.]

### Diagrammes des cas d’utilisation.

[UML n'impose ni ne préconise aucun format particulier de description textuelle des cas d'utilisation.

Un cas d'utilisation est composé de deux grandes parties : 1/ La description des interactions dans un cas typique de succès (le cas nominal) accompagné d'informations de contexte, 2/ les variations du cas (cas particuliers, extensions), contraintes diverses, questions ouvertes et 3/ diverses illustrations comme des diagrammes de séquence système par exemple. A ce sujet, Craig Larman (Larman, 2003) détaille également la notion de contrats d'opération système .

L'idée est de fournir un format de présentation textuelle à la fois souple et riche. En nous inspirant du livre d’Alistair Cockburn sur la question (CockBurn, 2003) , nous retiendrons le format suivant :]

|  |
| --- |
| **Cas d’utilisation : « Nom du cas d’utilisation »** *Utiliser une tournure à l’infinitif (ex : Réceptionner un colis).* |
| **Objectif :**  *Une description résumée permettant de comprendre l’intention principale du cas d’utilisation. Cette partie est souvent renseignée au début du projet dans la phase de découverte des cas d’utilisation.*  **Niveau d’objectif**  *Une valeur parmi {Stratégique, Objectif utilisateur, Sous-fonctionnalité}.*  **Evènement déclencheur :**  *Evènement déclencheur du cas d’utilisation*  **Acteurs principaux :**  *Ceux qui vont réaliser le cas d’utilisation (la relation avec le cas d’utilisation est illustrée par le trait liant le cas d’utilisation et l’acteur dans un diagramme de cas d’utilisation).*  **Acteurs secondaires (Parties prenantes et intérêts) :**  *Liste des parties prenantes et leurs intérêts.*  **Préconditions :**  *qui décrivent dans quel état doit être le système (l'application) avant que ce cas d'utilisation puisse être déclenché (Ex : un contrat existe avec le client)*  **Post conditions :**  *qui décrivent l'état du système à l'issue des différents scénarii (Ex : un contrat est créé et le système back-office est mis à jour avec le nouveau contrat créé)*  **Scénario nominal**   1. *Description de l’action.* 2. *Description de l’action.* 3. *……...* 4. *Description de l’action.*   **Scénario alternatif( ou d’extension)**  *2.a*  *2.b*  *…*  *4.a*  **Scénario d’exception**  *1*  *2.b*  *…*  *4.a*  **Résultat mesurable :**  *Le résultat final mesurable du cas d’utilisation.*  **Extensions du cas d’utilisation :**  *Occasionnellement un cas d’utilisation peut être découpé en plusieurs cas d’utilisation plus petits. Il y aura un cas d’utilisation principal qui utilise plusieurs autres cas d’utilisation. Cette section liste tous les cas d’utilisation que ce cas d’utilisation utilise.*   * *<numéro de l'étape> : <condition> : <sous-cas d'utilisation>* * *<numéro de l'étape> : <condition> : <sous-cas d'utilisation>*   **Contraintes :**  *Ce sont les contraintes non fonctionnelles, telles que temps de réponse, capacité de montée en charge, confidentialité, date de disponibilité, format de stockage des données…*  **Questions ouvertes :**  *Tout questionnement susceptible de pointer des zones d'ombre.*  **Annexe :**  *Illustrations et autres informations concernant le cas. On peut, ici, présenter une instance du cas d'utilisation (scénario) sous la forme d'un diagramme de séquence système (DSS) par exemple. Un DSS (Larman, 2003) est un diagramme de séquence de haut niveau représentant les interactions entre les acteurs externes et le système, ce dernier étant vu comme une boîte noire.* |

**Niveaux d'objectif**

Un cas d'utilisation, comme tout diagramme UML, permet de décrire une réalité selon différents niveaux de raffinement. Il convient, entre autres, de signaler le "niveau d'abstraction" de la vue afin de permettre au lecteur une meilleure interprétation de ce qui est et n'est pas montré. Concernant les cas d'utilisation, nous parlons alors de niveaux d'objectif.

Cockburn (Cockburn, 2003) définit trois niveaux d'objectif :

* **Niveau stratégique**

Présente le contexte général, les grandes fonctions du système (approche métier), ses objectifs.

Un cas d'utilisation de niveau stratégique implique plusieurs objectifs utilisateurs et s'étale généralement sur plusieurs jours, semaines, mois ou années.

Exemple : 1/ L'étudiant s'inscrit à une formation. 2/ L'étudiant s'inscrit à des modules 3/

L'étudiant passe un examen partiel 4/ L'étudiant obtient son diplôme.

* **Niveau objectif utilisateur**

C'est l'objectif suivi par un acteur en interaction avec le système. L'objectif utilisateur est celui qui représente le plus d'intérêt. Il correspond au processus métier élémentaire en ingénierie des processus métier (Cockburn, 2003). Sa durée, de 2 à 20 minutes, peut être réduite si le déclencheur est un système.

Exemple : Inscription à un module.

* **Niveau sous-fonctionnalité**

Ce sont des cas d'utilisation qui participent au bon déroulement de cas d'utilisation de niveau objectif utilisateur. Un cas d'utilisation de sous-fonctionnalité remplit un objectif partiel d'un cas d'utilisation d'objectif utilisateur, auquel il est lié par une relation d'inclusion (<<include>>).

Exemple : Identifier un utilisateur.

Exemple :

**Cas d'utilisation** : Communiquer des renseignements

**Acteur** : visiteur

**Parties prenantes et intérêts** :

* Visiteur : il veut communiquer ses coordonnées à l'établissement afin d'être contacté ultérieurement sur les formations qu'il aura sélectionnées.
* Service Administratif : il veut pouvoir contacter, en temps voulu, les personnes ayant manifesté un intérêt pour les formations disponibles dans l'établissement.

**Niveau** : Objectif utilisateur

**Portée** : Système Porte Ouverte

**Préconditions** : aucune

**Post-conditions** : Les données communiquées par le visiteur sont accessibles par le service administratif.

**Scénario nominal**

1. Le visiteur communique au système un choix de formation.

2. Le système communique un formulaire d'identification.

3. Le visiteur s'identifie.

4. Le système présente un formulaire correspondant à la formation choisie.

5. Le visiteur renseigne le formulaire et le soumet au système.

6. Le système enregistre les informations et signale le succès de l'opération.

**Extensions**

• \*. A tout moment : le visiteur peut abandonner l'opération en cours.

• 1a. Le visiteur est déjà authentifié : aller en 4

• 2a. Le visiteur n'est pas connu du système : sous-cas d'utilisation Création visiteur

• 5a. Certains champs obligatoires ne sont pas renseignés : aller en 4.

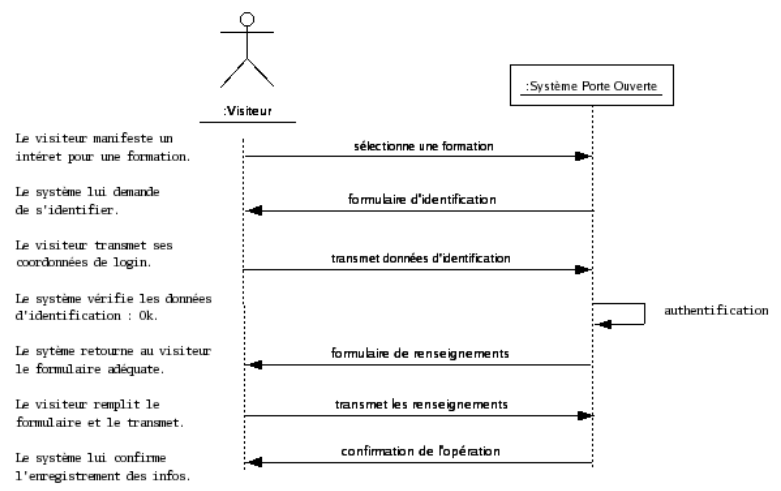
**Contraintes**

• Temps de réponse : La réception d'une réponse à l'identification (étape 4) ne doit pas dépasser 3 secondes.

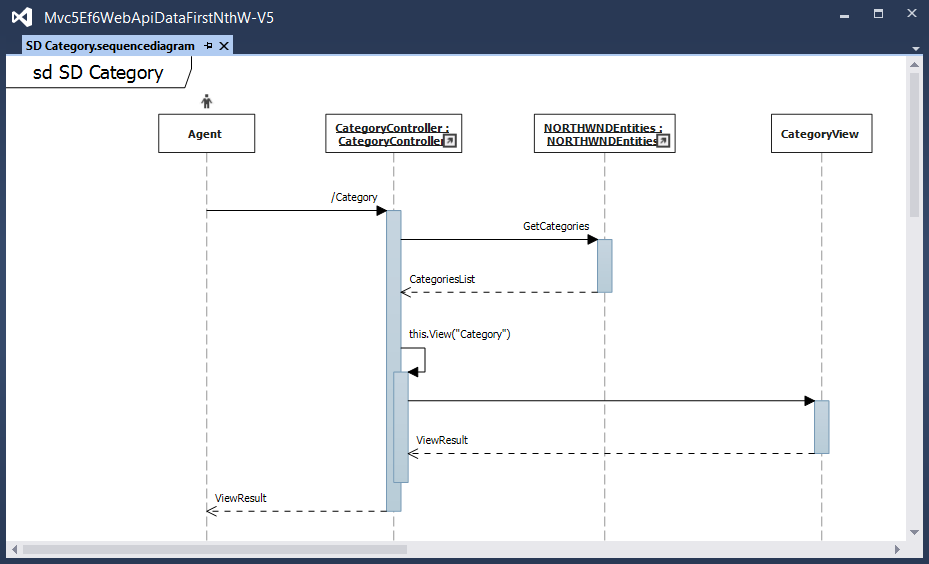
**Annexe**

Illustration du cas par un scénario représenté par un diagramme de séquence système (DSS).

Dans ce scénario, le visiteur est connu du système et n'abandonne pas l'opération.



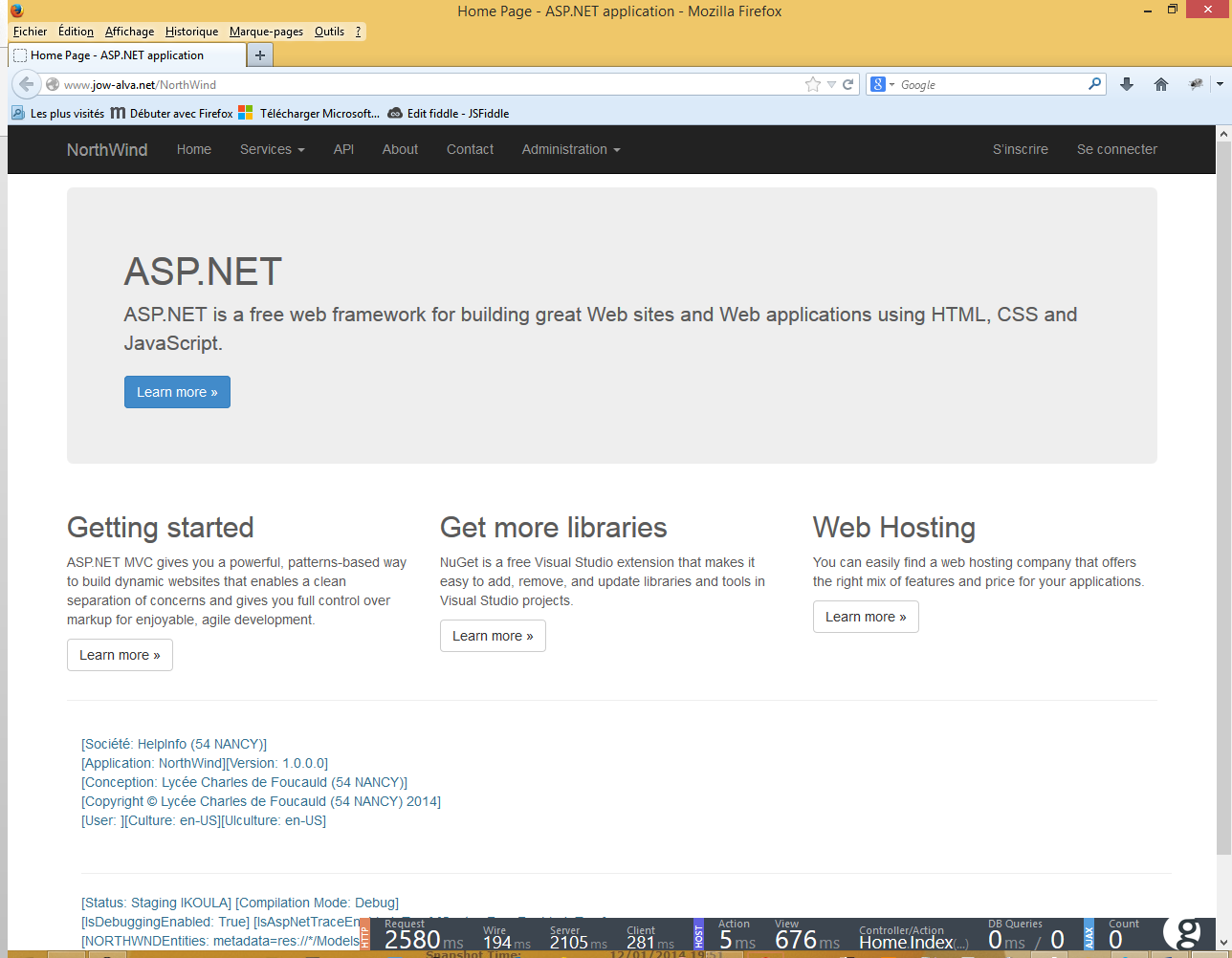
## Diagrammes de séquences ou/et de collaboration



Idem pour « Product », en modifiant « Category » par « Product ».

# PROTOTYPAGE DES I.H.M.

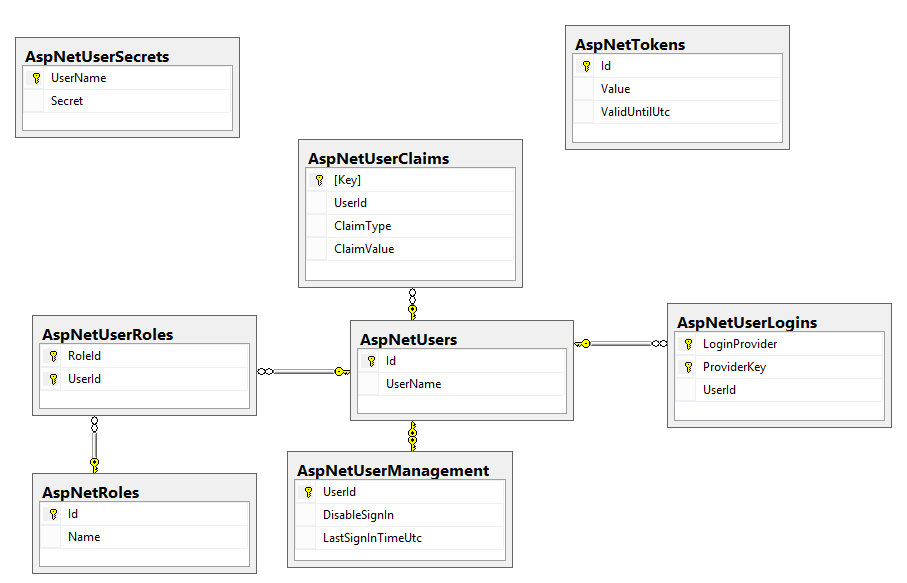
[Prototypage des IHM à partir des cas d’utilisation et des diagrammes de séquences.]



|  |  |
| --- | --- |
|  |  |

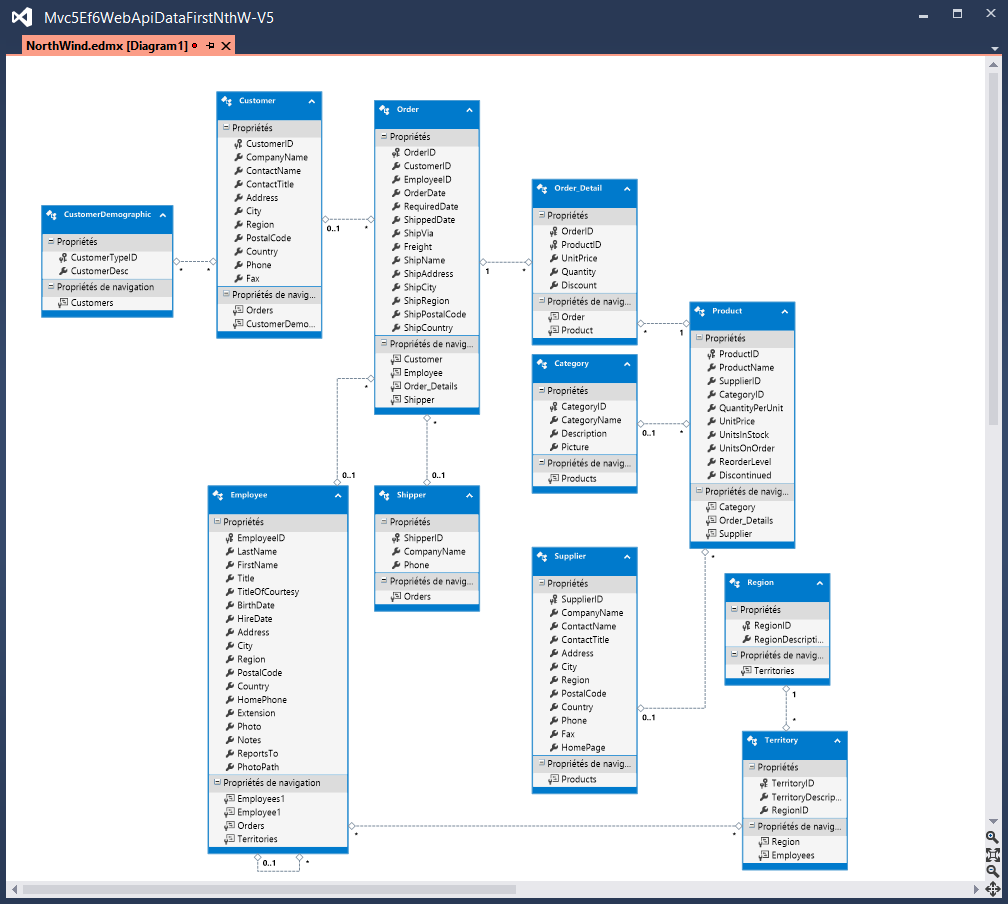
# Schéma des bases de données.

## ASP.NET Identity



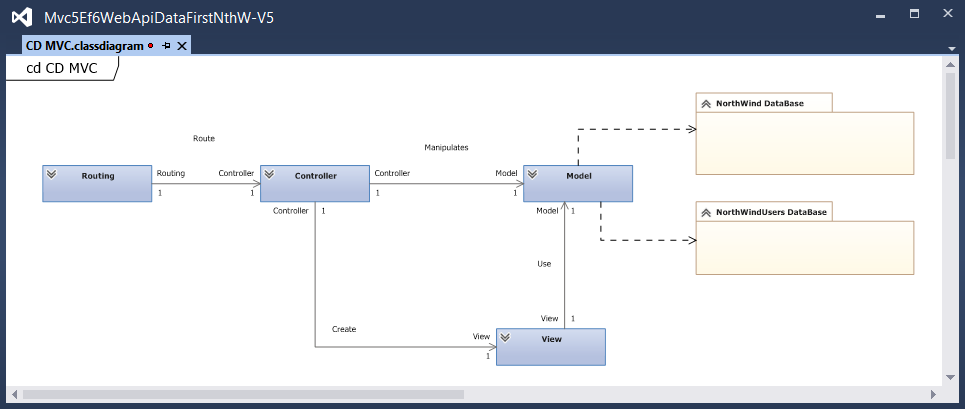
Commentaires :

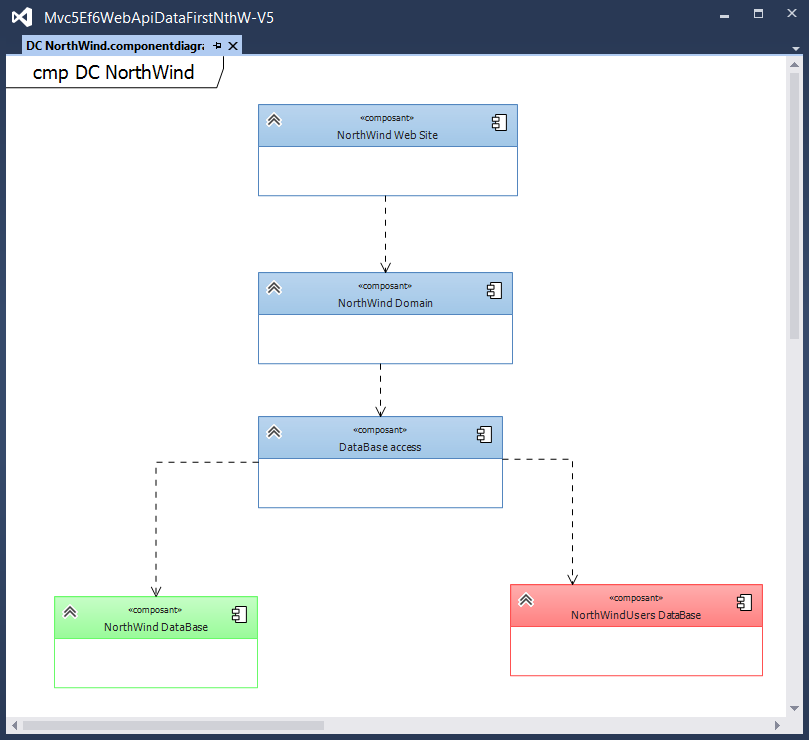
## NorthWind



Commentaires :

# Architecture logicielle





Processus d’exécution des applications MVC

Réf : <http://dotnet.developpez.com/mvc/mvc-understanding-execution-process/>

Les requêtes envoyées à une application Web basée sur ASP.NET MVC passent d'abord dans l'objet **UrlRoutingModule**, qui est un module HTTP. Ce module analyse la requête et effectue une sélection sur la route. L'objet **UrlRoutingModule** sélectionne la première route qui correspond à la demande actuelle. (Un objet route est une classe qui implémente **RouteBase**, et est généralement une instance de la classe Route). Si aucune route ne correspond, l'objet **UrlRoutingModule** ne fait rien et laisse la demande retourner au système de traitement standard des demandes d'ASP.NET ou d'IIS.

L'objet **UrlRoutingModule** obtient depuis l'objet Route l'objet **IRouteHandler** qui est associé à l'objet Route. Généralement, dans une application MVC, ce sera une instance de **MvcRouteHandler**. L'instance de **IRouteHandler** crée un objet **IHttpHandler** et le passe à l'objet **IHttpContext**. Par défaut, l'instance de **IHttpHandler** pour MVC est l'objet **MvcHandler**. L'objet **MvcHandler** sélectionne le contrôleur qui, à terme, traitera la demande.

L'**UrlRoutingModule** et l'**IHttpHandler** sont les points d'entrée du Framework ASP.NET MVC. Ils exécutent les actions suivantes :

* Sélectionner le contrôleur dans une application Web MVC.
* Obtenir une instance spécifique d'un contrôleur.
* Appeler la méthode Execute du contrôleur.

Le tableau suivant présente les étapes de l'exécution d'un projet Web MVC.

|  |  |
| --- | --- |
| Étape | Détails |
| Réception de la première requête de l'application | Dans le fichier Global.asax, les objets **Route** sont ajoutés à l'objet **RouteTable**. |
| Effectuer le routage | Le module **UrlRoutingModule** utilise le premier objet **Route** correspondant à la requête dans la collection **RouteTable** pour créer un objet **RouteData**, qu'il utilise ensuite pour créer un objet **RequestContext** (**IHttpContext**) objet. |
| Créer un gestionnaire de requête (request handler) MVC | Le module **MvcRouteHandler** crée une instance de la classe **MvcHandler**, et lui transmet l'instance de **RequestContext**. |
| Créer un contrôleur | L'objet **MvcHandler** utilise l'instance de **RequestContext** pour identifier l'objet **IControllerFactory** (généralement une instance de la classe **DefaultControllerFactory**) pour créer une instance de contrôleur. |
| Exécuter un contrôleur | L'instance de **MvcHandler** appelle la méthode **Execute** du contrôleur. |
| Invoquer une Action | La plupart des contrôleurs héritent de la classe de base de **Controller**. Pour les contrôleurs pour lesquels c'est le cas, l'objet **ControllerActionInvoker** qui est associé au contrôleur détermine quelle méthode d'action de la classe du contrôleur appeler, puis appelle la méthode. |
| Exécuter une fonction résultat | Une action typique de la méthode peut recevoir des entrées de l'utilisateur, préparer les données appropriées pour le contenu de la réponse, puis exécuter le résultat, en renvoyant un type **Result**. Les types de base de résultat qui peuvent être exécutés sont les suivants : **ViewResult** (qui rend une vue et qui est le type résultat le plus souvent utilisé), **RedirectToRouteResult**, **RedirectResult**, **ContentResult**, **JsonResult** et **EmptyResult**. |

# Architecture matérielle

# Planification des tâches

**Etat d’un élément** : Proposé🡺Accepté🡺Estimé🡺Prêt🡺En cours🡺Fini

## Tâches globales

## Détails des tâches

# Pré développement

## Mise en place de la solution sur le poste de développement

Création de l’arborescence sur le poste de développement

Modification: web.config

## Contrôle de code source

### Création des comptes sur Codeplex et Github

Codeplex [https://www.codeplex.com/

|  |  |  |
| --- | --- | --- |
| NOM Prénom | Username | Email |
|  |  |  |
| *M. José ALVAREZ* | jalvarez | jose.alvarez54@free.fr |
|  |  |  |
|  |  |  |
|  |  |  |

Github [https://github.com/]

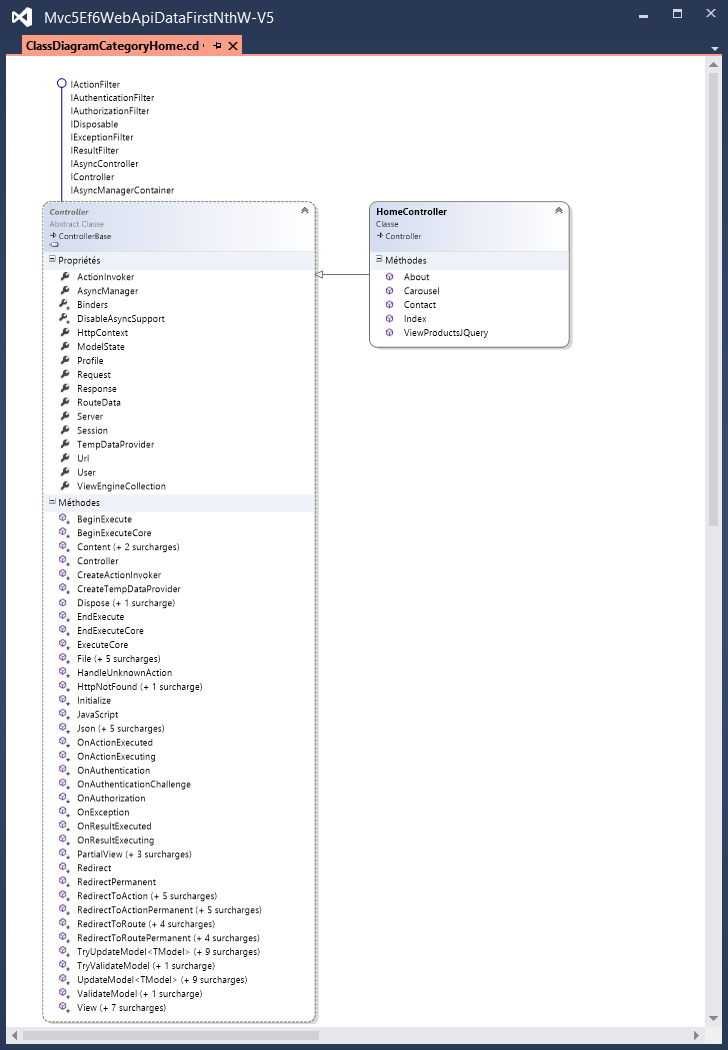
|  |  |  |
| --- | --- | --- |
| NOM Prénom | Username | Email |
|  |  |  |
| *M. José ALVAREZ* | jalvarez54 | jose.alvarez54@free.fr |
|  |  |  |
|  |  |  |
|  |  |  |

### Projet sur Codeplex et Visual Studio

Vidéo : [www.dailymotion.com/video/x19lsik\_mise-en-oeuvre-initiale-de-team-explorer-avec-codeplex\_tech](http://www.dailymotion.com/video/x19lsik_mise-en-oeuvre-initiale-de-team-explorer-avec-codeplex_tech)

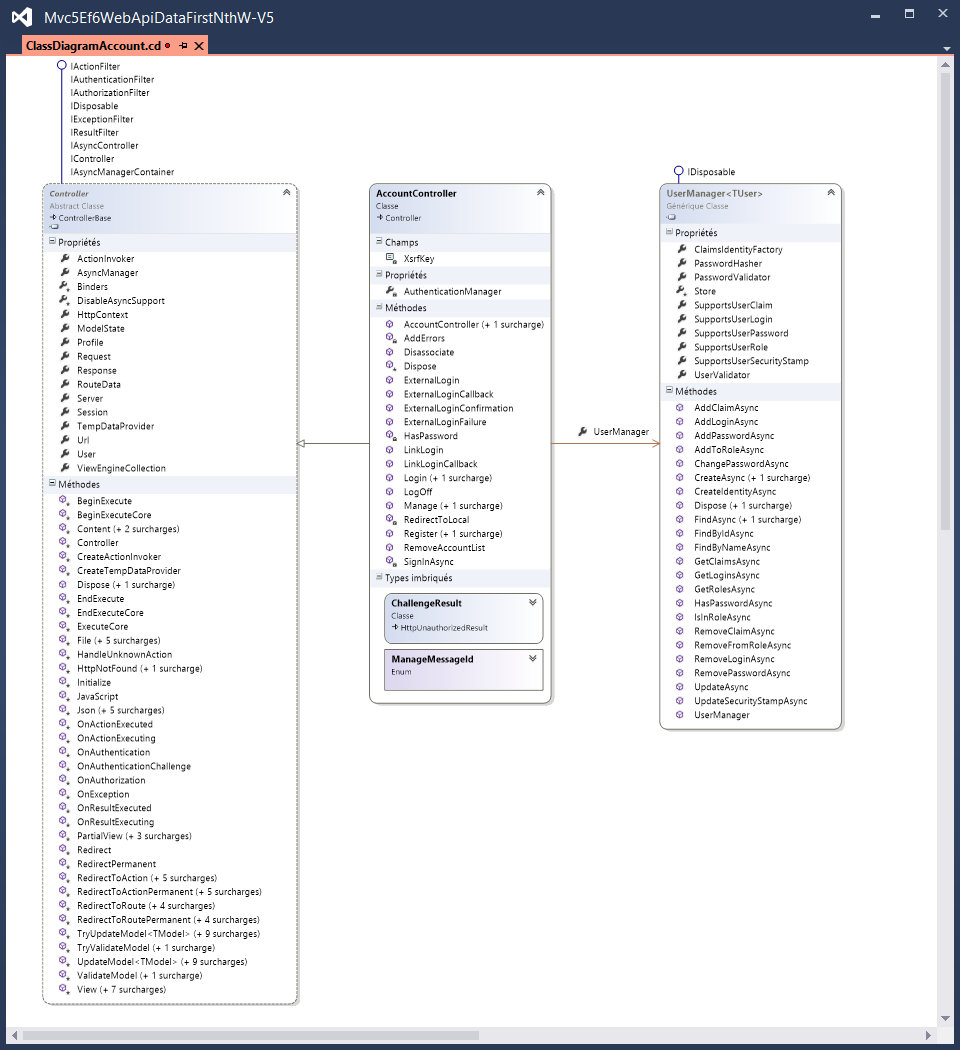
# Conception

## Home controller class diagram

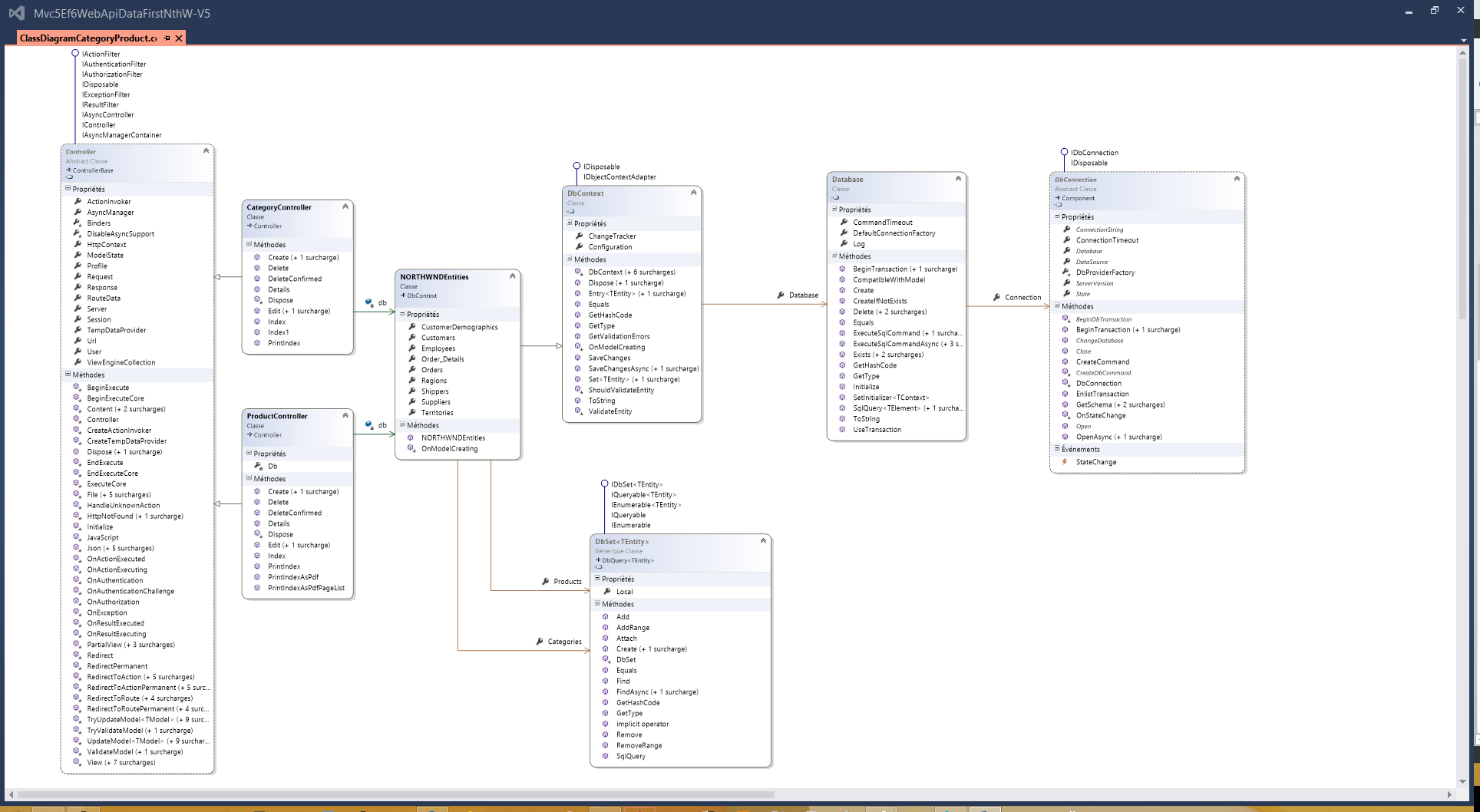


Idem pour « Product », en modifiant « Category » par « Product ».

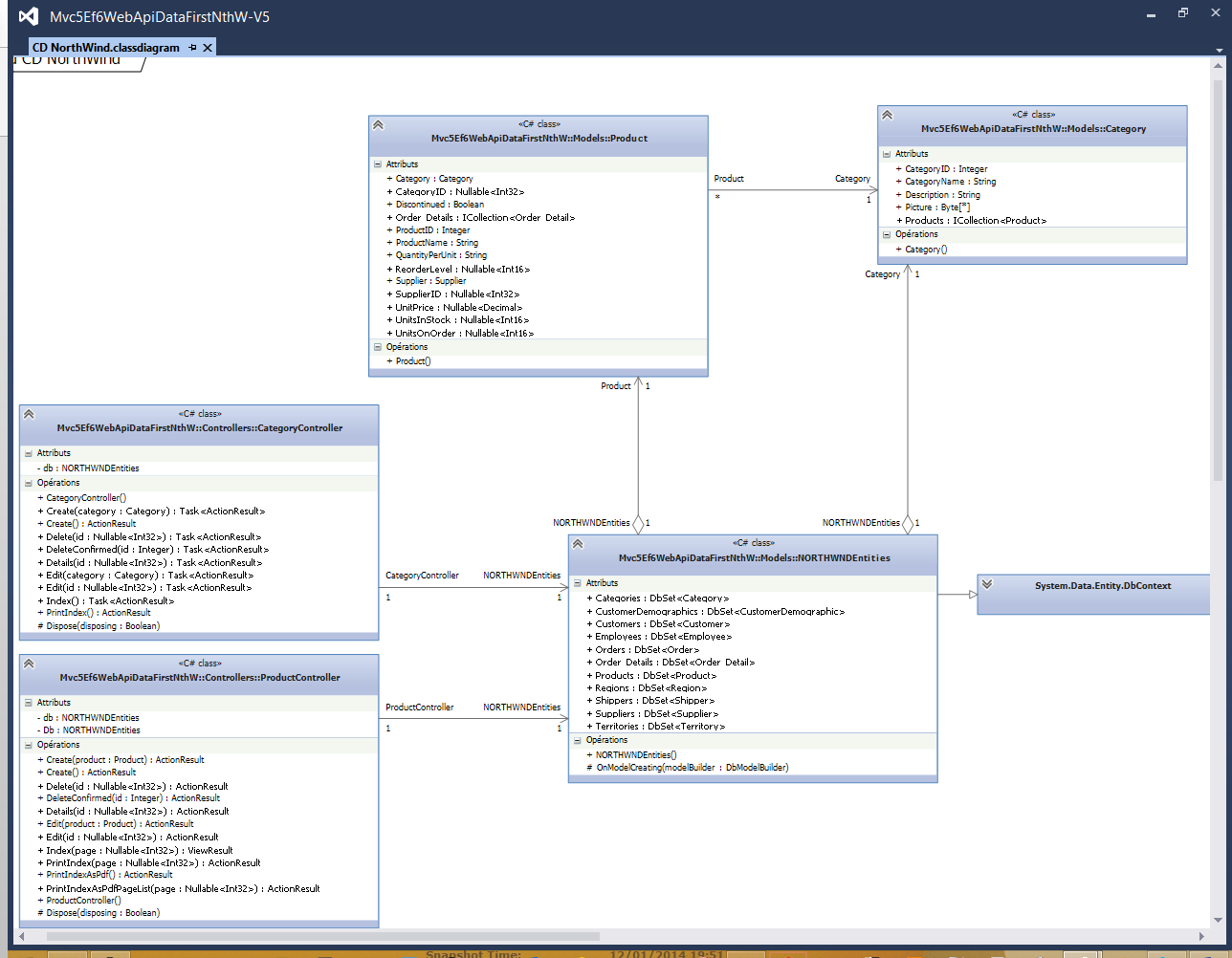
## Account Class Diagram.



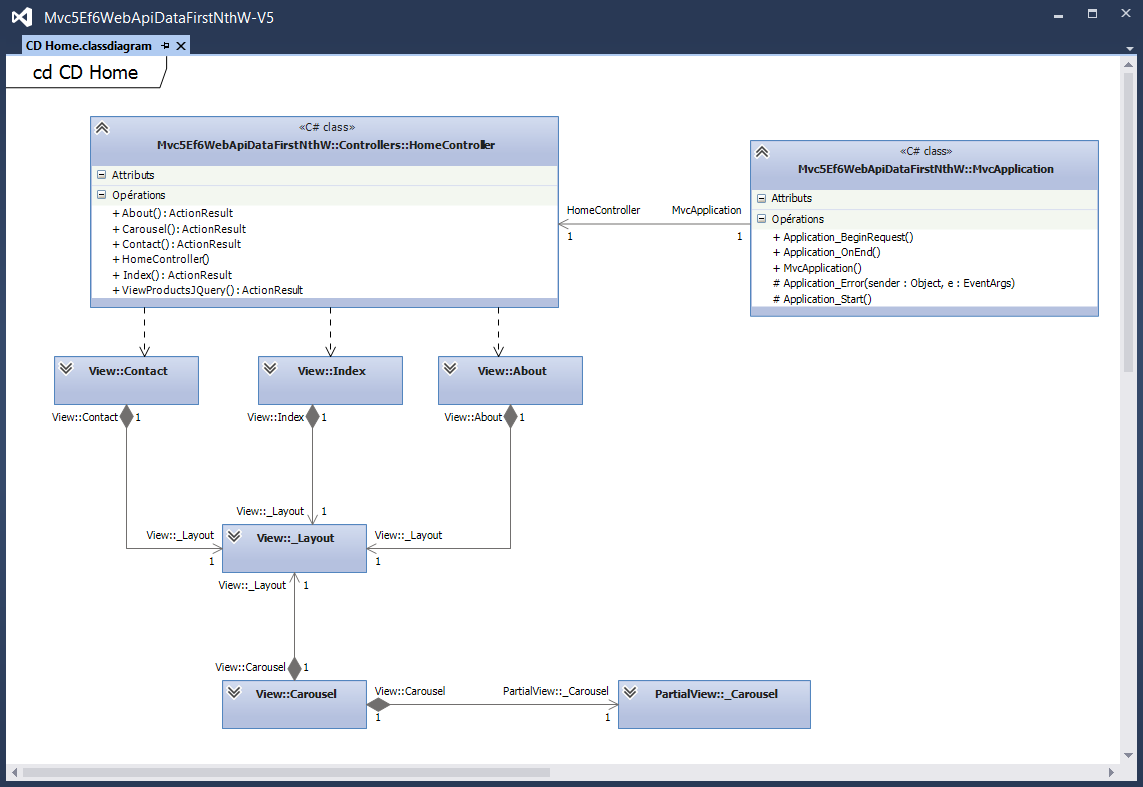
## Category/Product Class Diagram



## NorthWind class diagram



## HomeController class diagram



## Codage

### Internationalisation

Ref : <http://rdonfack.developpez.com/tutoriels/dotnet/internationalisation-application-asp-net-mvc-3-razor/>

#### Mécanismes

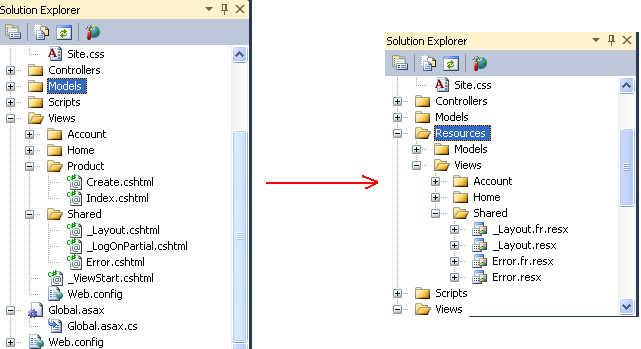
Lorsqu'on parle d'internationalisation d'une application sur la plateforme .NET, cela implique la mise en œuvre de deux mécanismes principaux à savoir : la globalisation et la localisation.

La **globalisation** est le processus de conception et de développement d'un produit logiciel adapté à plusieurs cultures. Une application globalisée prend en charge des interfaces utilisateurs localisées et des données régionales pour tous les utilisateurs. Par exemple, le format de la date en zone francophone est jj/mm/aaaa. Dans les pays anglophones, nous avons plutôt le format mm/jj/aaaa. La globalisation va consister à afficher le bon format dans une page en fonction de la langue du navigateur d'un utilisateur ou des options choisies par celui-ci.

La **localisation** est le processus d'adaptation de votre application à une culture et aux paramètres régionaux donnés. Vous devez pendant cette étape, personnaliser votre application pour les cultures auxquelles vous la destinez. Cette étape doit consister essentiellement en une traduction de l'interface utilisateur vers les langues cibles.

#### Architecture

Pour rester dans la logique de regroupement qu'impose ASP.NET MVC, il est préférable que l'arborescence de vos fichiers de ressources soit semblable à la capture ci-dessous :



|  |
| --- |
| @using Mvc5Ef6WebApiDataFirstNthW.Resources.Views.Shared; |

#### Modification des messages

|  |
| --- |
| <li>@Html.ActionLink(\_Layout.Products, "Index", "Product")</li> |

#### Localisation des messages d'erreur de validation et DisplayAttribute

|  |
| --- |
| [Required(ErrorMessageResourceName = "Required", ErrorMessageResourceType = typeof(AccountViewModels))]  [StringLength(100, ErrorMessageResourceName = "PasswordMinLength", ErrorMessageResourceType = typeof(AccountViewModels), MinimumLength = 6)]  [DataType(DataType.Password)]  [Display(Name = "Password", ResourceType = typeof(AccountViewModels))]  public string Password { get; set; } |

### Trace et logs

#### Centralisation de l’écriture des traces et des logs

Dans /Helpers/ MyTracer.cs, création d’une classe statique MyTracer exposant une méthode statique MyTrace, afin d’uniformiser les messages.

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW.Helpers  {      public static class MyTracer      {          public static void MyTrace(TraceLevel type, Type theClass, string controllerName, string actionName, string message, Exception e)          {              string messageToShow;              log4net.ILog log;              log = log4net.LogManager.GetLogger(theClass);              if (e != null)              {                  messageToShow = string.Format("\tClass={0} \r\n\tController= {1} \r\n\tAction= {2} \r\n\tMessage= {3} \r\n\tGetBaseException= {4}",                              theClass.ToString(), controllerName, actionName, message, e.GetBaseException().ToString());              }              else              {                  messageToShow = string.Format("\tClass={0} \r\n\tController= {1} \r\n\tAction= {2} \r\n\tMessage= {3}",                              theClass.ToString(), controllerName, actionName, message);              }              switch (type)              {                  case TraceLevel.Info:                      Trace.TraceInformation(messageToShow);                      log.Info(messageToShow);                      break;                  case TraceLevel.Error:                      Trace.TraceError(messageToShow);                      log.Error(messageToShow);                      break;                  case TraceLevel.Warning:                      Trace.TraceWarning(messageToShow);                      log.Warn(messageToShow);                      break;              }          }      }  } |

Nb : Cas particulier pour Elmah (cf. §xxxxxxx), ce dernier ne capture que les exceptions non gérées (unhandled exceptions). Aussi pour qu’il prenne en compte les exceptions gérées, il faut lui signaler ces exceptions :

|  |
| --- |
| Elmah.ErrorSignal.FromCurrentContext().Raise(ex); //ELMAH Signaling |

D’où l’utilisation de la méthode statique SignalExceptionToElmahAndTrace dans /Helpers/Utils pour les « Actions » gérant les exceptions fatales ou non. Cf. § 10.6.2.3 (Pour les tests)

|  |
| --- |
| public static void SignalExceptionToElmahAndTrace(Exception ex, System.Web.Mvc.Controller lui)  {      Mvc5Ef6WebApiDataFirstNthW.Controllers.MyErrorController mec = lui as Mvc5Ef6WebApiDataFirstNthW.Controllers.MyErrorController;      MyTracer.MyTrace(          System.Diagnostics.TraceLevel.Error,          mec.GetType(),          mec.ControllerContext.RouteData.Values["controller"].ToString(),          mec.ControllerContext.RouteData.Values["action"].ToString(),          ex.Message, ex);        Elmah.ErrorSignal.FromCurrentContext().Raise(ex); //ELMAH Signaling  } |

Exemple d’utilisation dans un try… catch

|  |
| --- |
| Helpers.Utils.SignalExceptionToElmahAndTrace(ex, this); |

#### Filtre

Afin de centraliser le traçage du passage dans toutes les actions, on implémente un filtre « MyRequestLogFilter » dérivant d’IActionFilter. La méthode OnActionExecuting sera exécutée lors de l’exécution d’une action quelconque.

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW.CustomFiltersAttributes  {      public class MyRequestLogFilter : IActionFilter      {          public void OnActionExecuting(ActionExecutingContext filterContext)          {              // Don't show filter multiple times when using Html.RenderAction or Html.Action.              if (filterContext.IsChildAction == true)                  return;              // Action trace centralyzed              Helpers.MyTracer.MyTrace(System.Diagnostics.TraceLevel.Info, this.GetType(),               filterContext.Controller.ToString(), filterContext.ActionDescriptor.ActionName, "Passage dans une Action", null);          }  } |

[TODO] Ajouter filtre Authentification

* <http://www.dotnetcurry.com/showarticle.aspx?ID=957>

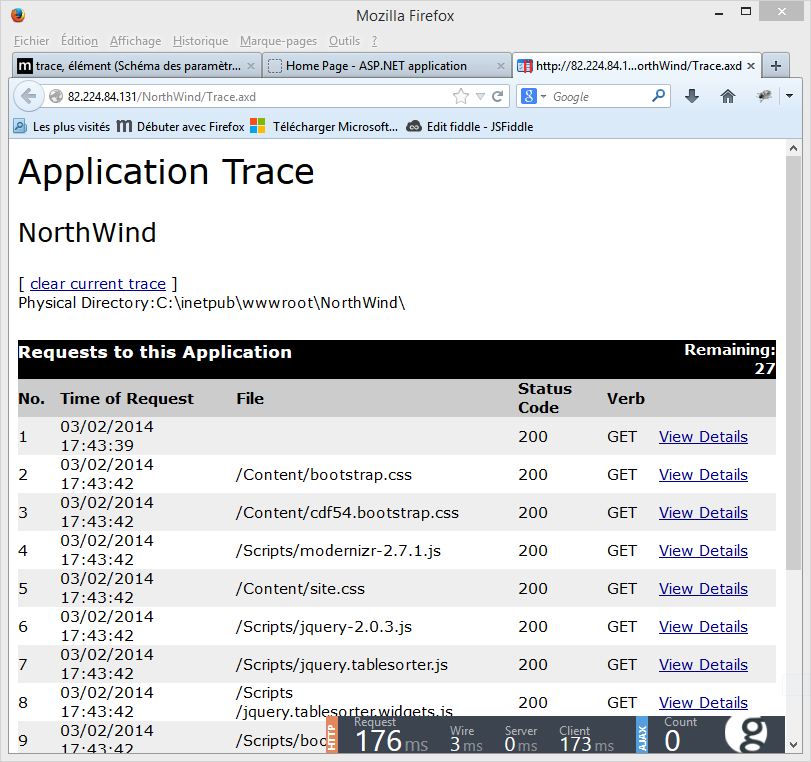
#### Trace

Ref : <http://msdn.microsoft.com/fr-fr/library/6915t83k%28v=vs.85%29.aspx>

L'élément trace de Web.Config, configure le service de traçage du code d'ASP.NET qui contrôle la collecte, le stockage et l'affichage des résultats de traçage.

|  |
| --- |
| <system.web>      <trace enabled="false" requestLimit="40" localOnly="false" />    </system.web> |

Lorsque le traçage est activé, chaque demande de page génère des messages de trace qui peuvent être ajoutés à la sortie de page ou stockés dans un journal des traces de l'application. La visionneuse de traçage ASP.NET (**Trace.axd**) permet de consulter le contenu du journal des traces. Par défaut, Trace.axd est ajouté à l'élément httpHandlers.



#### System.Diagnostics.Trace

Réf : <http://msdn.microsoft.com/en-us/library/xe8whywc%28v=vs.110%29.aspx>

Paramètrage :

|  |
| --- |
| <system.diagnostics>    <!--CDF54-Begin-->      <trace autoflush="true" indentsize="4">        <listeners>          <add name="myTextListener" type="System.Diagnostics.TextWriterTraceListener" initializeData="App\_Data/SysDiagTextWriterOutput.txt" />        </listeners>      </trace>    <!--CDF54-End-->    </system.diagnostics> |

Utilisation :

|  |
| --- |
| using System.Diagnostics;  Trace.TraceInformation(messageToShow);  Trace.TraceError(messageToShow);  Trace.TraceWarning(messageToShow); |

#### Elmah

Réf :

* <http://code.google.com/p/elmah/>
* <http://www.codeproject.com/Articles/516626/Elmah>

#### Log4Net

Réf :

* <http://logging.apache.org/log4net/>
* <http://www.codeproject.com/Articles/140911/log4net-Tutorial>
* <http://lutecefalco.developpez.com/tutoriels/dotnet/log4net/introduction/>
* <http://www.andyfrench.info/2012/03/xml-schema-for-log4net.html>

### Gestion des performances avec Glimpse

Réf :

* <http://getglimpse.com/>
* <http://glimpse.codeplex.com/>

### Gestion des exceptions

#### Elément « customErrors » de Web.Config

L’attribut « **mode** » de l’élément « **customErrors** », spécifie si les erreurs personnalisées sont activées, désactivées ou visibles uniquement pour les clients distants.

Cet attribut peut avoir l'une des valeurs suivantes :

* **On**: Spécifie que les erreurs personnalisées sont activées. Si aucun attribut defaultRedirect n'est spécifié, les utilisateurs voient s'afficher une erreur générique. Les erreurs personnalisées sont montrées aux clients distants et à l'hôte local.
* **Off**: Spécifie que les erreurs personnalisées sont désactivées. Les erreurs ASP.NET détaillées sont montrées aux clients distants et à l'hôte local.
* **RemoteOnly**: Spécifie que les erreurs personnalisées ne sont visibles que pour les clients distants et les erreurs ASP.NET sont montrées à l'hôte local. Valeur par défaut.

La valeur par défaut est RemoteOnly.

|  |
| --- |
| <!--RemoteOnly, On, Off.-->      <customErrors mode="RemoteOnly" defaultRedirect="~/MyError/UnHandledException">        <error statusCode="404" redirect="~/MyError/NotFound" />        <error statusCode="403" redirect="~/MyError/AccessDenied" />      </customErrors> |

Mode : On ou Off pour le développement et RemoteOnly pour la production.

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW  {      public class FilterConfig      {          public static void RegisterGlobalFilters(GlobalFilterCollection filters)          {              //filters.Add(new HandleErrorAttribute());              filters.Add(new MyRequestLogFilter());          }      }  } |

#### filters.Add(new HandleErrorAttribute());

Ce « filter-attribut » est ajouté par défaut dans « **RegisterGlobalFilters** », il s’applique donc à l’ensemble de l’application sans avoir à décorer une action par une annotation « **[HandleError]** ».

Ce handle utilise par défaut la View « **Error.cshtml** » qui doit être enregistrée dans « **View/Shared** ». Il ne détecte que les exceptions ayant lieu dans le code et non les erreurs http : 403, 404…

D’où la nécessité de déclarer dans l’élément « **customErrors** » de **Web.Config** le controller et l’action à exécuter en cas d’erreur 403 ou 404.

Un autre inconvénient de ce handle, est qu’il ne permet pas d’effectuer de traçage.

Nous commentons donc ce filtre et utilisons une autre route par défaut que « **/Views/Shared/Error.cshtml** » soit « **~/MyError/UnHandledException** », où on va pouvoir tracer l’exception :

#### Cas particulier des exceptions gérées

Pour les exceptions gérées, fatales ou non nous devons dans le catch du try-catch, tracer les exceptions et signaler à Elmah cette exception, dans la mesure où ce dernier ne trace que les exceptions non gérées.

* Pour les exceptions non fatales, nous devons dans le catch tracer l’exception et la signaler à Elmah, puis poursuivre l’exécution du code.
* Pour les exceptions fatales, nous devons dans le catch tracer l’exception et la signaler à Elmah, puis afficher une page d’erreur à l’utilisateur.

Nous utilisons pour cela une méthode static dans « /Helpers/Utils » :

|  |
| --- |
| public static void SignalExceptionToElmahAndTrace(Exception ex, System.Web.Mvc.Controller lui)  {      MyTracer.MyTrace(          System.Diagnostics.TraceLevel.Error,          mec.GetType(),          mec.ControllerContext.RouteData.Values["controller"].ToString(),          mec.ControllerContext.RouteData.Values["action"].ToString(),          ex.Message, ex);      Elmah.ErrorSignal.FromCurrentContext().Raise(ex); //ELMAH Signaling  } |

Exemple d’utilisation pour un exception non fatale et une exception fatale :

|  |
| --- |
| /// <summary>  /// Show SignalExceptionToElmahAndTrace usage  with try...catch in a non fatal handled exception  /// </summary>  /// <returns></returns>  public ActionResult ForTestHandledException()  {      try      {          throw new ApplicationException("Handled exception for test");      }      catch (Exception ex)      {  **Helpers.Utils.SignalExceptionToElmahAndTrace(ex, this);**      }      // ...............................      // Continu program...      // ...............................      // Finally call the view      return View();  } |
| /// <summary>  /// Show SignalExceptionToElmahAndTrace usage with try...catch in a fatal handled exception  /// </summary>  /// <returns></returns>  public ActionResult ForTestHandledFatalException()  {      try      {          throw new ApplicationException("Fatal error for test");      }      catch (Exception ex)      {  **Helpers.Utils.SignalExceptionToElmahAndTrace(ex, this);**  **// RedirectToAction called if fatal error**  **return RedirectToAction("HandledFatalException", "MyError", new { area = "" });**      }      // View called if no fatal error      return View(); // Never reached  } |

Remarquons dans ce dernier cas l’utilisation de « RedirectToAction » qui redirige vers l’action « /MyError/HandledFatalException » qui demandera l’affichage de sa vue « HandledFatalException.cshtml » on mentionne l’Area racine afin que cette redirection puisse être effectuée dans une autre Area..

#### « Controller » MyError

|  |
| --- |
| /// <summary>  /// Handle 404 error  /// </summary>  /// <param name="aspxerrorpath"></param>  /// <returns>ViewResult</returns>  public ActionResult NotFound(string aspxerrorpath)  {      ViewData["error\_path"] = aspxerrorpath;      MyTracer.MyTrace(TraceLevel.Warning, this.GetType(), null, null, string.Format("404 resource {0} not found", aspxerrorpath), null);        return View();  }  /// <summary>  /// Handle 403 error  /// </summary>  /// <param name="action"></param>  /// <returns>ViewResult</returns>  public ActionResult AccessDenied(string action)  {      @ViewData["action"] = action;      MyTracer.MyTrace(TraceLevel.Warning, null, null, null, string.Format("Acess denied in the action {0}", action), null);      return View();  }  /// <summary>  /// Handle erors # 403, 404  /// </summary>  /// <returns>ViewResult</returns>  public ActionResult UnHandledException(string aspxerrorpath)  {      ViewData["error\_path"] = aspxerrorpath;      string actionName = this.ControllerContext.RouteData.Values["action"].ToString();      string controllerName = this.ControllerContext.RouteData.Values["controller"].ToString();      Exception unHandledException = new ApplicationException("My UnHandled Exception");      MyTracer.MyTrace(TraceLevel.Error, this.GetType(), controllerName, actionName, string.Format("Unhandled exception in {0}", aspxerrorpath), unHandledException);      return View(new HandleErrorInfo(unHandledException, controllerName, actionName));  }  /// <summary>  /// For handled fatal exceptions  /// </summary>  /// <returns></returns>  public ActionResult HandledFatalException()  {      return View();  } |

#### « View » correspondant aux actions

Le contenu est très simple à comprendre, donc non commenté ici.

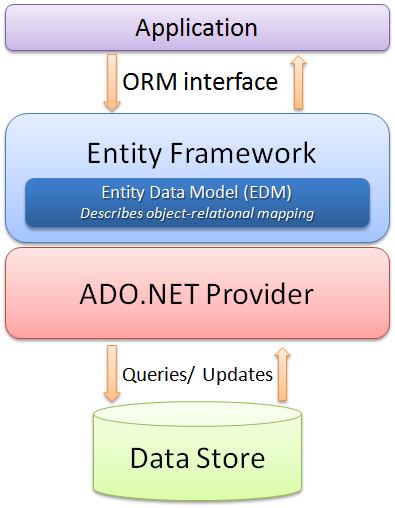
#### Pour les tests :

|  |
| --- |
| #region Test errors actions  public ActionResult NoView()  {      return View();  }  public ActionResult ForTestUnHandledException()  {      throw new NotImplementedException();  }  /// <summary>  /// Show SignalExceptionToElmahAndTrace usage  with try...catch in a non fatal handled exception  /// </summary>  /// <returns></returns>  public ActionResult ForTestHandledException()  {      try      {          throw new ApplicationException("Handled exception for test");      }      catch (Exception ex)      {          Helpers.Utils.SignalExceptionToElmahAndTrace(ex, this);      }      // ...............................      // Continu program...      // ...............................      // Finally call the view      return View();  }  /// <summary>  /// Show SignalExceptionToElmahAndTrace usage with try...catch in a fatal handled exception  /// </summary>  /// <returns></returns>  public ActionResult ForTestHandledFatalException()  {      try      {          throw new ApplicationException("Fatal error for test");      }      catch (Exception ex)      {          Helpers.Utils.SignalExceptionToElmahAndTrace(ex, this);          // RedirectToAction called if fatal error          return RedirectToAction("HandledFatalException", "MyError", new { area = "" });      }      // View called if no fatal error      return View(); // Never reached  }  #endregion |

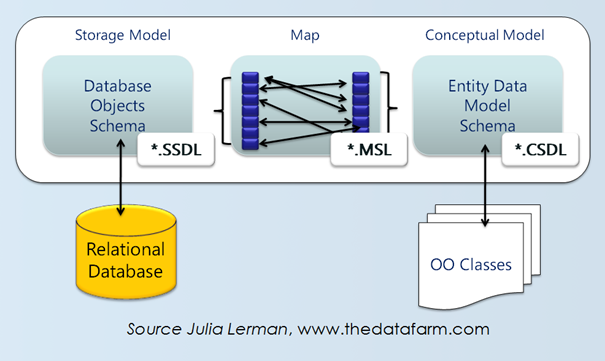
### « Category » et « Product »

L’accès aux données repose sur **l’ORM** (Object Relational Mapping) Entity Framework dans sa version 6.

L’Entity Framework a été conçu pour permettre aux développeurs de créer des applications d'accès aux données en programmant par rapport à un modèle d'application conceptuel au lieu de programmer directement par rapport à un schéma de stockage relationnel. L'objectif est de diminuer la quantité de code et les besoins en maintenance pour les applications orientées objet.



Visual Studio dispose d’un outil (**EDM Designer** : Entity Data Model Designer)) générant 3 modèles à partir d’une base de données existante.



Vocabulaire et principaux objets d’EF.

**Entity :**

Une entité, est un objet mappé généralement sur une table physique de la base de données.

**Objet DbContext :**

C’est l'objet principal de l'interaction avec une base de données en utilisant un modèle spécifique.

DbContext effectue le mapping de l’ORM entre les classes entités et les tables de la base de données.

Il est responsable de la connexion et de la déconnexion à la base de données.

Il est responsable du suivi des modifications des entités.

Pour la persistance, il utilise ses méthodes « SaveChanges » ou « SaveChangesAsync » pour pousser dans la base de données les entités en fonction de leur état.

**Objet DbSet :**

Un objet DbSet représente la collection de toutes les entités dans le contexte, ou des entités pouvant être interrogées à partir de la base de données, d'un type donné. Les objets DbSet sont créés à partir d'un DbContext à l'aide de la méthode DbContext.Set. Ils exposent les méthodes Add, Attach, Create, Find, Remove… Pour manipuler l’entité dans le context.

Il effectue les opérations **CRUD** sur un type spécifique de modèle. Il implémente pour cela les interfaces, IQueryable<TEntity>, IEnumerable<TEntity>, IQueryable, IEnumerable, permettant notamment à LINQ de manipuler ces entités.

Exemple : « db.Categories.ToListAsync() », où « **db** » est une instance de DbContex » et « Categories » une instance de la collection « DbSet<Category> »

**ConnectionString :**

Le constructeur de DbContext, passe à sa classe de base le nom de la ConnexionString, dont les paramètres sont enregistrés dans Web.Config, afin que cette dernière puisse créer une instance du provider à partir des informations : « data source » et « initial catalog ».

|  |
| --- |
| <connectionStrings>      <add name="DefaultConnection"          connectionString="Data Source=.\SQLEXPRESS;           Initial Catalog=aspnet-NORTHWND;           Integrated Security=True"          providerName="System.Data.SqlClient" />      <add name="NORTHWNDEntities"           connectionString="metadata=res://\*/Models.NorthWind.csdl|res://\*/Models.NorthWind.ssdl|res://\*/Models.NorthWind.msl;           provider=System.Data.SqlClient;           provider connection string=’           data source=.\SQLEXPRESS;           initial catalog=NORTHWND;           integrated security=True;           MultipleActiveResultSets=True;           App=EntityFramework’"           providerName="System.Data.EntityClient" />    </connectionStrings> |

Avec quelques adaptations, idem pour « Products » en remplaçant « Category » par « Product ».

#### « Controller »

|  |
| --- |
| public class CategoryController : Controller  {      private NORTHWNDEntities db = new NORTHWNDEntities();        public ActionResult IndexToPdf()      {          return new Rotativa.ActionAsPdf(                         "Index")                         {                             FileName = "Categories.pdf",                             CustomSwitches = "--print-media-type",                             PageOrientation = Rotativa.Options.Orientation.Landscape                         };      }      public ActionResult PrintIndex()      {          return new Rotativa.ActionAsPdf(                         "Index") { FileName = "Categories.pdf" };      }        // GET: /Category/      [OutputCache(Duration=30)]      public async Task<ActionResult> Index()      {          return View(await db.Categories.ToListAsync());      }        // GET: /Category/      [OutputCache(Duration = 30)]      public async Task<ActionResult> TrunckedCategories()      {          //return View(await db.Categories.ToListAsync());            var model = await (from b in db.Categories                             orderby b.CategoryName                             select new CategoryViewModel {                                 CategoryName = b.CategoryName,                                 Description = b.Description                             }).ToListAsync();            return View(model);      }        // GET: /Category/Details/5      public async Task<ActionResult> Details(int? id)      {          if (id == null)          {              return new HttpStatusCodeResult(HttpStatusCode.BadRequest);          }          Category category = await db.Categories.FindAsync(id);          if (category == null)          {              return HttpNotFound();          }          return View(category);      }      protected override void Dispose(bool disposing)      {          if (disposing)          {              db.Dispose();          }          base.Dispose(disposing);      }  } |

Le « **controller** » passe à la « **view** » le « **model** » contenant les données à afficher, la « **view** » retour un « **ViewResult** » au « **controller** », qui retournera au client ce flux html au navigateur.

#### « Model »

Nous utilisons une approche « **Data Fir**st », le modèle « **NorthWind.edmx** » a été généré par Visual Studio 2013.

Il comprend les classes suivantes :

|  |
| --- |
| public partial class NORTHWNDEntities : DbContext  public partial class Category  public partial class Product  Etc… |

|  |
| --- |
| public partial class NORTHWNDEntities : DbContext  {      public NORTHWNDEntities()          : base("name=NORTHWNDEntities")      {      }        protected override void OnModelCreating(DbModelBuilder modelBuilder)      {          throw new UnintentionalCodeFirstException();      }        public virtual DbSet<Category> Categories { get; set; }      public virtual DbSet<CustomerDemographic> CustomerDemographics { get; set; }      public virtual DbSet<Customer> Customers { get; set; }      public virtual DbSet<Employee> Employees { get; set; }      public virtual DbSet<Order\_Detail> Order\_Details { get; set; }      public virtual DbSet<Order> Orders { get; set; }      public virtual DbSet<Product> Products { get; set; }      public virtual DbSet<Region> Regions { get; set; }      public virtual DbSet<Shipper> Shippers { get; set; }      public virtual DbSet<Supplier> Suppliers { get; set; }      public virtual DbSet<Territory> Territories { get; set; }  } |

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW.Models  {      using System;      using System.Collections.Generic;        public partial class Category      {          public Category()          {              this.Products = new HashSet<Product>();          }            public int CategoryID { get; set; }          public string CategoryName { get; set; }          public string Description { get; set; }          public byte[] Picture { get; set; }            public virtual ICollection<Product> Products { get; set; }      }  } |

Notez que ces classes sont « **partial** », et que l’on peut donc les étendre, il n’est en effet pas question de modifier le code généré par Visual Studio 2013.

#### « View »

|  |
| --- |
| @model IEnumerable<Mvc5Ef6WebApiDataFirstNthW.Models.Category>  @using Mvc5Ef6WebApiDataFirstNthW.Helpers    @{      ViewBag.Title = "Categories";  }  @section PrintToPdf  {      @Html.ActionLink("PDF", ViewContext.RouteData.GetRequiredString("action")+"ToPdf", ViewContext.RouteData.GetRequiredString("controller"))  }    @section styles      {      <style type="text/css">          @@media print {              footer, #pNotPrinted, #buttonNotPrinted {                  display: none;              }                #h3PageBreak {                  page-break-before: always;              }                @@page {                  margin: 0cm;              }                header:after {                  content: "jow-alva.net/NorthWind";              }                header:before {                  display: block;                  content: "Thank you for printing.";                  margin-bottom: 10px;                  border: 1px solid #bbb;                  padding: 3px 5px;                  font-style: italic;              }          }      </style>  }  <header></header>    <h2>@ViewBag.Title</h2>  <h3>Responsive table</h3>  <div class="table-responsive, container-fluid">      <table class="table table-striped table-bordered table-hover table-condensed">          <tr>              <th>                  @Html.DisplayNameFor(model => model.CategoryName)              </th>              <th>                  @Html.DisplayNameFor(model => model.Description)              </th>              <th>                  @Html.DisplayNameFor(model => model.Picture)              </th>              @\*<th></th>\*@          </tr>          @foreach (var item in Model)          {              <tr>                  <td data-title=@Html.DisplayNameFor(model => model.CategoryName)>                      @Html.DisplayFor(modelItem => item.CategoryName)                  </td>                  <td data-title=@Html.DisplayNameFor(model => model.Description)>                      @Html.DisplayFor(modelItem => item.Description)                  </td>                  <td data-title=@Html.DisplayNameFor(model => model.Picture)>                      @MyRazorHelpers.TagImage("img-responsive", item.Picture, @item.CategoryName, @item.CategoryName, "0")                  </td>              </tr>          }      </table>  </div>    <h3 id="h3PageBreak">Responsive table plus No More Table</h3>  <section class="container-fluid" id="no-more-tables" >      <table class="table-bordered table-striped table-condensed cf">            <thead class="cf">              <tr>                  <th>                      @Html.DisplayNameFor(model => model.CategoryName)                  </th>                  <th>                      @Html.DisplayNameFor(model => model.Description)                  </th>                  <th>                      @Html.DisplayNameFor(model => model.Picture)                  </th>                  @\*<th></th>\*@              </tr>          </thead>          <tbody>              @foreach (var item in Model)              {                  <tr>                      <td data-title=@Html.DisplayNameFor(model => model.CategoryName)>                          @Html.DisplayFor(modelItem => item.CategoryName)                      </td>                      <td data-title=@Html.DisplayNameFor(model => model.Description)>                          @Html.DisplayFor(modelItem => item.Description)                      </td>                      @{byte[] photo = item.Picture;                      string imageSrc = null;                      if (photo != null)                      {                          MemoryStream ms = new MemoryStream();                          ms.Write(photo, 78, photo.Length - 78); // strip out 78 byte OLE header (don't need to do this for normal images)                          string imageBase64 = Convert.ToBase64String(ms.ToArray());                          imageSrc = string.Format("data:image/png;base64,{0}", imageBase64);                      }                      }                      <td data-title=@Html.DisplayNameFor(model => model.Picture)>                          <img class="img-responsive" src="@imageSrc" title="@item.CategoryName" alt="@item.CategoryName" border="0" />                      </td>                  </tr>              }          </tbody>      </table>  </section> |

#### « ViewModel »

L’« **action** » « **TrunckedCategories**» » ci-dessous, aura besoin de communiquer des informations à sa vue qui ne correspondent pas proprement à nos objets modèles. Ici « **Picture** » n’est pas récupéré. Nous ne voulons pas modifier nos modèles pour que cela convienne à nos vues ; les classes modèles devraient représenter notre domaine, et non l'interface utilisateur. La solution adoptée, est d'utiliser le **pattern ViewModel**. Lorsque l'on utilise ce pattern, nous créons des classes fortement typées qui sont optimisées pour les scénarios de notre vue spécifique, et qui exposent les propriétés pour les valeurs/contenus dynamiques dont la vue à besoin. Nos classes contrôleur peuvent remplir et transmettre ces classes optimisées à nos vues afin de les utiliser. Cela permet un type sécurisé, lors de la compilation, et la présence de l'IntelliSense dans les vues.

|  |
| --- |
| public async Task<ActionResult> TrunckedCategories()  {      var model = await (from b in db.Categories                         orderby b.CategoryName                         select new CategoryViewModel {                             CategoryName = b.CategoryName,                             Description = b.Description                         }).ToListAsync();        return View(model);  } |

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW.Models  {      public class CategoryViewModel      {          public string CategoryName { get; set; }          public string Description { get; set; }      }  } |

#### « TrunckedCategories View »

|  |
| --- |
| @model IEnumerable<Mvc5Ef6WebApiDataFirstNthW.Models.CategoryViewModel>    @{      ViewBag.Title = "Categories";  }    <h2>Categories without pictures</h2>    <table class="table">      <tr>          <th>              @Html.DisplayNameFor(model => model.CategoryName)          </th>          <th>              @Html.DisplayNameFor(model => model.Description)          </th>      </tr>    @foreach (var item in Model) {      <tr>          <td>              @Html.DisplayFor(modelItem => item.CategoryName)          </td>          <td>              @Html.DisplayFor(modelItem => item.Description)          </td>      </tr>  }  </table> |

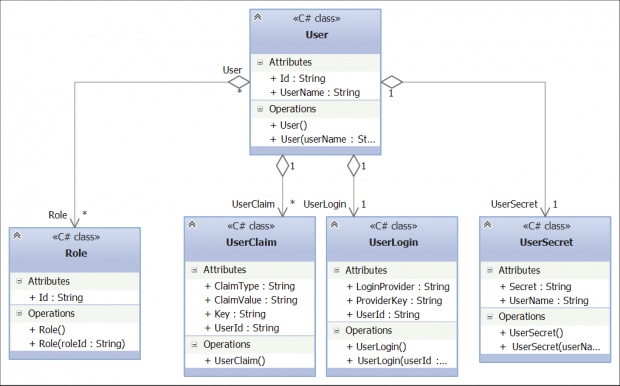
### ASP.NET Identity & Extension

#### Microsoft ASP.NET Identity

Réf :

* <http://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity>
* <http://www.asp.net/identity/overview/getting-started/adding-aspnet-identity-to-an-empty-or-existing-web-forms-project>
* <http://blogs.infinitesquare.com/b/beedoo/archives/aspnet-identity-et-personnalisation-du-modele-de-stockage-des-utilisateurs#.UvXUaoXVo9A>
* <http://dotnet.dzone.com/articles/look-new-identity-system>
* <http://www.novanet.no/blog/olav-nybo/dates/2013/8/forms-authentication-using-aspnet-identity--vs-2013-preview/>

Le model



Comme le montre le schéma ci-dessus le modèle est constitué d'un « User » et les objets associés suivants:

UserSecret

Pour la certification des formes ce sera le mot de passe haché

UserLogin

Le loginProvider, pour l'authentification par formulaire ce sera "Local". UserLogin a également la clé utilisée pour identifier l'utilisateur avec le fournisseur de connexion.

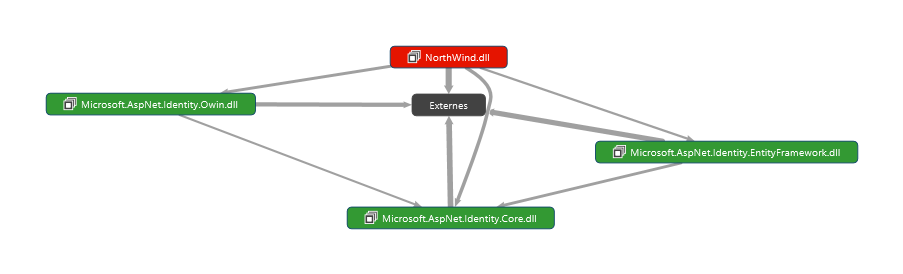
UserClaim

Une liste des « Claims » de l'utilisateur

Role

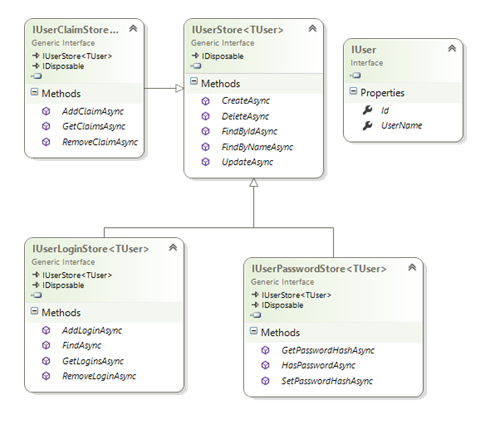
Une liste des rôles auxquels l'utilisateur a

Assembly faisant partie d’ASP.NET Identity



Microsoft.AspNet.Identity.Core

Logique de gestion des utilisateurs et le rôle de base



IUserStore

Offre les opérations de base pour créer, supprimer, mettre à jour et récupérer un utilisateur :

IUserPasswordStore

Offre les opérations pour permettre la gestion de mots de passe pour vos utilisateurs :

IUserClaimStore

Offre les opérations nécessaires à la gestion des différents claims que l’on souhaite associer à un utilisateur. Typiquement, la notion de profil telle qu’on pouvait la connaître avec les ASP.NET Membership tend à disparaître pour être remplacée par des identités basées sur des claims, beaucoup plus génériques / flexibles.

IUserLoginStore

ASP.NET Identity intègre directement la notion multi logins pour la gestion des utilisateurs, permettant d’offrir des scénarios où un utilisateur peut choisir de se connecter avec un compte local, par exemple, puis associer ses différents comptes Facebook, Microsoft, Google… Cette interface vous permet d’implémenter les différentes opérations nécessaires à la mise en place de ce scénario.

IUserSecurityStampStore

Offre des opérations nécessaires à la génération de hash de sécurité qui dépendent des différents paramètres de connexion d’un utilisateur (différents logins et providers, notamment).

Microsoft.AspNet.Identity.EntityFramework

Persistance des utilisateurs, rôles, claims etc. basé sur EnitityFramework.



La classe **ApplicationUser** représente la définition de ce que doit être un utilisateur dans votre application. Elle dérive directement d’une classe **IdentityUser** définie dans le package EntityFramework d’ASP.NET Identity :



La classe **UserManager** est définie dans le package IdentityCore et utilise le **UserStore** défini dans le package Entity Framework Identity. C’est cette dernière classe qui implémente les différentes interfaces de store décrites plus haut.

NB : Ce modèle étant basé sur Entity Framework code first, il est tout à fait possible de modifier, en utilisant les Db code migrations, par exemple.

#### Extension des comptes Identity et implémentation des rôles

Objectif : L’objectif que l’on se fixe ici, est de transformer le code fourni par le Template de Visual Studio 2013 afin de :

1. Faire en sorte que seul l’« **Administrateur** » puisse créer des comptes, donc de supprimer la possibilité de s’inscrire.
2. D’avoir à notre disposition quatre rôles : « Administrateur », « Développeur », « Gestionnaire » et « Invité ».
3. Créer une page permettant de lister les utilisateurs
4. Créer une page permettant de positionner un utilisateur dans un ou plusieurs rôles.
5. Avoir la possibilité de supprimer, de modifier un profil utilisateur

Nous nous inspirons du code de « John Atten » :  
<http://www.codeproject.com/Articles/682113/Extending-Identity-Accounts-and-Implementing-Role>

Nous utilisons une seule base de données ; les tables d’ASP.NET Identity seront donc incluses dans la base de l’application NorthWind « NORTHWND.mdf ».

##### ConnectionString

|  |
| --- |
| <add name="DefaultConnection"          connectionString="data source=.\SQLEXPRESS;          initial catalog=NORTHWND;          Persist Security Info=True;          User ID=cdf54projet;          Password=p@ssword"          providerName="System.Data.SqlClient" /> |
| <add name="NORTHWNDEntities"          connectionString="metadata=res://\*/Models.NorthWind.csdl|res://\*/Models.NorthWind.ssdl|res://\*/Models.NorthWind.msl;provider=System.Data.SqlClient;          provider=System.Data.SqlClient;          provider connection string=&quot;          Data Source=.\SQLEXPRESS;          Initial Catalog=NORTHWND;          User ID=cdf54projet;          Password=p@ssword;          MultipleActiveResultSets=True;          App=EntityFramework&quot;"          providerName="System.Data.EntityClient" /> |

##### Configuration de la « Migration » Entity FrameWork

Commande Enable-Migrations

|  |
| --- |
| PM> **Enable-Migrations -ContextTypeName Mvc5Ef6WebApiDataFirstNthW.Models.ApplicationDbContext –EnableAutomaticMigrations**  Vérification du ciblage par le contexte d'une base de données existante…  Fonctionnalité Migrations Code First activée pour le projet Mvc5Ef6WebApiDataFirstNthW. |

Cette commande a ajouté un dossier Migrations à notre projet. Ce nouveau dossier contient une classe Configuration. Cette classe nous permet de configurer le comportement de Migrations pour notre contexte.



Étant donné qu'il n'y a deux contextes dans notre projet (un de type **CodeFirst**, et un de type **DataFirst**), nous devons renseigner la commande sur le type de contexte auquel cette configuration s'applique ici « **ApplicationDbContext** ».

Voici le code par défaut dans la classe de Configuration des Migrations :

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW.Migrations  {      using Mvc5Ef6WebApiDataFirstNthW.Models;      using System;      using System.Data.Entity;      using System.Data.Entity.Migrations;      using System.Linq;            internal sealed class Configuration              : DbMigrationsConfiguration<Mvc5Ef6WebApiDataFirstNthW.Models.ApplicationDbContext>          {              public Configuration()              {                  AutomaticMigrationsEnabled = true;              }              protected override void Seed(                  Mvc5Ef6WebApiDataFirstNthW.Models.ApplicationDbContext context)              {                  //  This method will be called after migrating to the latest version.                    //  You can use the DbSet<T>.AddOrUpdate() helper extension method                  //  to avoid creating duplicate seed data. E.g.                  //                  //    context.People.AddOrUpdate(                  //      p => p.FullName,                  //      new Person { FullName = "Andrew Peters" },                  //      new Person { FullName = "Brice Lambson" },                  //      new Person { FullName = "Rowan Miller" }                  //    );                  //              }          }      }  } |

Nous devrons l’adapter à notre application. Nous voulons effectivement créer les tables dans la base existante, les rôles et ajouter des utilisateurs. Il nous faut avant tout, écrire le code permettant la gestion des utilisateurs et des rôles.

##### Suppression des fichiers et du code non utilisés par l’application

Simplification de AccountController

Suppression des méthodes :

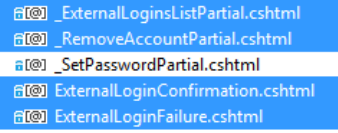
* Dissociate()
* ExternalLogin()
* ExternalLoginCallback()
* LinkLogin()
* LinkLoginCallback()
* ExternalLoginConfirmation()
* ExternalLoginFailure()
* RemoveAccountList()

Ainsi que les éléments :

* La constante membre XsrfKey
* Toute la classe ChallengeResult

Puis cliquer droit dans le code pour supprimer les « using » non utilisés.

Suppression des « Views » non utilisés



Nettoyage des « Views »

**Login.cshtml** - Suppression de l’option de connexions aux réseaux sociaux.

Toute la <div> contenant la <section> « socialLoginForm ».

**Manage.cshtml** - Suppression de l’option de connexions Externes.

La <section> « externalLogins».

Suppression des « Models » non utilisés

**AccountViewModels.cs** – Suppression de la classe « ExternalLoginConfirmationViewModel ».

##### Extension des « Models » et des « View Models »

Les classes « Model » utilisées par l’application pour gérer « Identity » et autorisation sont contenues dans le fichier « **IdentityModels.cs** ».

|  |
| --- |
| // You can add profile data for the user by adding more properties to your ApplicationUser class, please visit http://go.microsoft.com/fwlink/?LinkID=317594 to learn more.  public class ApplicationUser : IdentityUser  {  }    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  {      public ApplicationDbContext()          : base("DefaultConnection")      {      }  } |
| Nous ajoutons les propriétés imposées par l’application |
| public class ApplicationUser : IdentityUser  {      [Required]      public string FirstName { get; set; }      [Required]      public string LastName { get; set; }      [Required]      public string Email { get; set; }  } |

Nous ajoutons la classe « **IdentityManager** » qui permet la gestion des utilisateurs et des rôles.

|  |
| --- |
| public class IdentityManager  {      public bool RoleExists(string name)      {          var rm = new RoleManager<IdentityRole>(              new RoleStore<IdentityRole>(new ApplicationDbContext()));          return rm.RoleExists(name);      }      public bool CreateRole(string name)      {          var rm = new RoleManager<IdentityRole>(              new RoleStore<IdentityRole>(new ApplicationDbContext()));          var idResult = rm.Create(new IdentityRole(name));          return idResult.Succeeded;      }      public bool CreateUser(ApplicationUser user, string password)      {          var um = new UserManager<ApplicationUser>(              new UserStore<ApplicationUser>(new ApplicationDbContext()));          var idResult = um.Create(user, password);          return idResult.Succeeded;      }      public bool AddUserToRole(string userId, string roleName)      {          var um = new UserManager<ApplicationUser>(              new UserStore<ApplicationUser>(new ApplicationDbContext()));          var idResult = um.AddToRole(userId, roleName);          return idResult.Succeeded;      }      public void ClearUserRoles(string userId)      {          var um = new UserManager<ApplicationUser>(              new UserStore<ApplicationUser>(new ApplicationDbContext()));          var user = um.FindById(userId);          var currentRoles = new List<IdentityUserRole>();          currentRoles.AddRange(user.Roles);          foreach (var role in currentRoles)          {              um.RemoveFromRole(userId, role.Role.Name);          }      }  } |

Extension du « ViewModel » Account

Dans la classe « RegisterViewModel » ajout de :

|  |
| --- |
| // New Fields added to extend Application User class:    [Required]  [Display(Name = "First Name")]  public string FirstName { get; set; }  [Required]  [Display(Name = "Last Name")]  public string LastName { get; set; }  [Required]  public string Email { get; set; }  // Return a pre-poulated instance of AppliationUser:  public ApplicationUser GetUser()  {      var user = new ApplicationUser()      {          UserName = this.UserName,          FirstName = this.FirstName,          LastName = this.LastName,          Email = this.Email,      };      return user;  } |

Ajout de la classe « EditUserViewModel ».

|  |
| --- |
| public class EditUserViewModel  {      public EditUserViewModel() { }        // Allow Initialization with an instance of ApplicationUser:      public EditUserViewModel(ApplicationUser user)      {          this.UserName = user.UserName;          this.FirstName = user.FirstName;          this.LastName = user.LastName;          this.Email = user.Email;      }        [Required]      [Display(Name = "User Name")]      public string UserName { get; set; }        [Required]      [Display(Name = "First Name")]      public string FirstName { get; set; }        [Required]      [Display(Name = "Last Name")]      public string LastName { get; set; }        [Required]      public string Email { get; set; }        // Return a pre-populated instance of AppliationUser:      public ApplicationUser GetUser()      {          var user = new ApplicationUser()          {              UserName = this.UserName,              FirstName = this.FirstName,              LastName = this.LastName,              Email = this.Email,          };          return user;      }    } |

Ajout de la classe « SelectRoleEditorViewModel».

|  |
| --- |
| // SelectRoleEditorViewModel, Revisited:  // Used to display a single role with a checkbox, within a list structure:  public class SelectRoleEditorViewModel  {      public SelectRoleEditorViewModel() { }      public SelectRoleEditorViewModel(IdentityRole role)      {          this.RoleName = role.Name;      }        public bool Selected { get; set; }        [Required]      public string RoleName { get; set; }  } |

Ajout de la classe « SelectUserRolesViewModel».

|  |
| --- |
| // SelectUserRolesViewModel – Revisited:  public class SelectUserRolesViewModel  {      public SelectUserRolesViewModel()      {          this.Roles = new List<SelectRoleEditorViewModel>();      }        // Enable initialization with an instance of ApplicationUser:      public SelectUserRolesViewModel(ApplicationUser user)          : this()      {          this.UserName = user.UserName;          this.FirstName = user.FirstName;          this.LastName = user.LastName;            var Db = new ApplicationDbContext();            // Add all available roles to the list of EditorViewModels:          var allRoles = Db.Roles;          foreach (var role in allRoles)          {              // An EditorViewModel will be used by Editor Template:              var rvm = new SelectRoleEditorViewModel(role);              this.Roles.Add(rvm);          }            // Set the Selected property to true for those roles for          // which the current user is a member:          foreach (var userRole in user.Roles)          {              var checkUserRole =                  this.Roles.Find(r => r.RoleName == userRole.Role.Name);              checkUserRole.Selected = true;          }      }        public string UserName { get; set; }      public string FirstName { get; set; }      public string LastName { get; set; }      public List<SelectRoleEditorViewModel> Roles { get; set; }  } |

##### Extension du « Controller » Account.

Le « Controller » du Template propose les « Actions » suivantes :

* **Register** (Essentiellement pour créer un utilisateur)
* **Manage** (Essentiellement pour permettre à un utilisateur de changer son mot de passe)
* **Login**
* **LogOff**

Modification de la méthode « Register » de « AccountControllre » (Création d’utilisateur)

|  |
| --- |
| //Modified Register Method  //  // POST: /Account/Register  [HttpPost]  //[AllowAnonymous]  [Authorize(Roles = "Admin")]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Register(RegisterViewModel model)  {      if (ModelState.IsValid)      {          // Ajout          var user = model.GetUser();            // Modifie          //var user = new ApplicationUser() { UserName = model.UserName };          var result = await UserManager.CreateAsync(user, model.Password);            if (result.Succeeded)          {              var idManager = new IdentityManager();              idManager.AddUserToRole(user.Id, "User");              //await SignInAsync(user, isPersistent: false);              //return RedirectToAction("Index", "Home");              return RedirectToAction("Index", "Account");          }          else          {              AddErrors(result);          }      }        // Si nous sommes arrivés là, un échec s’est produit. Réafficher le formulaire      return View(model);  } |

Ajout de la méthode « Index » à « AccountController » (Vue Liste des utilisateurs)

|  |
| --- |
| // New Index Method:  [Authorize(Roles = "Admin")]  public ActionResult Index()  {      var Db = new ApplicationDbContext();      var users = Db.Users;      var model = new List<EditUserViewModel>();      foreach (var user in users)      {          var u = new EditUserViewModel(user);          model.Add(u);      }      return View(model);  } |

Ajout de la méthode « Edit » à « AccountController »

|  |
| --- |
| // New Edit Method:  [Authorize(Roles = "Admin")]  public ActionResult Edit(string id, ManageMessageId? Message = null)  {      var Db = new ApplicationDbContext();      var user = Db.Users.First(u => u.UserName == id);      var model = new EditUserViewModel(user);      ViewBag.MessageId = Message;      return View(model);  }  [HttpPost]  [Authorize(Roles = "Admin")]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Edit(EditUserViewModel model)  {      var Db = new ApplicationDbContext();      var user = Db.Users.First(u => u.UserName == model.UserName);      if (ModelState.IsValid)      {          // Update the user data:          user.FirstName = model.FirstName;          user.LastName = model.LastName;          user.Email = model.Email;          Db.Entry(user).State = System.Data.Entity.EntityState.Modified;          await Db.SaveChangesAsync();          return RedirectToAction("Index");      }      model.PhotoUrl = user.PhotoUrl;      // If we got this far, something failed, redisplay form      return View(model);  } |

Ajout de la méthode « Delete » à « AccountController »

|  |
| --- |
| // New Delete Methods:  [Authorize(Roles = "Admin")]  public ActionResult Delete(string id = null)  {      var Db = new ApplicationDbContext();      var user = Db.Users.First(u => u.UserName == id);      var model = new EditUserViewModel(user);      if (user == null)      {          return HttpNotFound();      }      return View(model);  }  [HttpPost, ActionName("Delete")]  [ValidateAntiForgeryToken]  [Authorize(Roles = "Admin")]  public ActionResult DeleteConfirmed(string id)  {      var Db = new ApplicationDbContext();      var user = Db.Users.First(u => u.UserName == id);      Db.Users.Remove(user);      Db.SaveChanges();      return RedirectToAction("Index");  } |

Ajout de la méthode « UserRoles » à « AccountController »

|  |
| --- |
| // New UserRoles Method(s):  [Authorize(Roles = "Admin")]  public ActionResult UserRoles(string id)  {      var Db = new ApplicationDbContext();      var user = Db.Users.First(u => u.UserName == id);      var model = new SelectUserRolesViewModel(user);      return View(model);  }  [HttpPost]  [Authorize(Roles = "Admin")]  [ValidateAntiForgeryToken]  public ActionResult UserRoles(SelectUserRolesViewModel model)  {      if (ModelState.IsValid)      {          var idManager = new IdentityManager();          var Db = new ApplicationDbContext();          var user = Db.Users.First(u => u.UserName == model.UserName);          idManager.ClearUserRoles(user.Id);          foreach (var role in model.Roles)          {              if (role.Selected)              {                  idManager.AddUserToRole(user.Id, role.RoleName);              }          }          return RedirectToAction("Index");      }      return View();  } |

##### Basiques « Views » pour le management d’« Identity » basé sur les rôles.

Modification de Register.cshtml

|  |
| --- |
| //Add the LastName, FirstName, Email and photo Properties:  <div class="form-group">      @Html.LabelFor(m => m.LastName, new { @class = "col-md-2 control-label" })      <div class="col-md-10">          @Html.TextBoxFor(m => m.LastName, new { @class = "form-control" })      </div>  </div>  <div class="form-group">      @Html.LabelFor(m => m.FirstName, new { @class = "col-md-2 control-label" })      <div class="col-md-10">          @Html.TextBoxFor(m => m.FirstName, new { @class = "form-control" })      </div>  </div>  <div class="form-group">      @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })      <div class="col-md-10">          @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })      </div>  </div> |

Ajout des « Views ».

Edit, Delete et Index

Modification de Index

Remplacer

|  |
| --- |
| <td>      @Html.ActionLink("Edit", "Edit", new { /\* id=item.PrimaryKey \*/}) |      @Html.ActionLink("Details", " Details", new { /\* id=item.PrimaryKey \*/}) |      @Html.ActionLink("Delete", "Delete", new { /\* id=item.PrimaryKey \*/})  </td> |

Par

|  |
| --- |
| <td>      @Html.ActionLink("Edit", "Edit", new { id = item.UserName }) |      @Html.ActionLink("Roles", "UserRoles", new { id = item.UserName }) |      @Html.ActionLink("Delete", "Delete", new { id = item.UserName })  </td> |

Création de la « View » « UserRoles ».

Ajout d’une « View » vide MVC 5.

|  |
| --- |
| @model Mvc5Ef6WebApiDataFirstNthW.Models.SelectUserRolesViewModel    @{      ViewBag.Title = "UserRoles";  }    <h2> UserRoles </h2> |

Puis du code suivant

|  |
| --- |
| @model Mvc5Ef6WebApiDataFirstNthW.Models.SelectUserRolesViewModel    @{      ViewBag.Title = "UserRoles";  }    <h2>Roles for user @Html.DisplayFor(model => model.UserName)</h2>  <hr />  @using (Html.BeginForm("UserRoles", "Account", FormMethod.Post, new { encType = "multipart/form-data", name = "myform" }))  {      @Html.AntiForgeryToken()        <div class="form-horizontal">          @Html.ValidationSummary(true)          <div class="form-group">              <div class="col-md-10">                  @Html.HiddenFor(model => model.UserName)              </div>          </div>          <h4>Select Role Assignments</h4>          <br />          <hr />          <table>              <tr>                  <th>                      Select                  </th>                  <th>                      Role                  </th>              </tr>              @Html.EditorFor(model => model.Roles)          </table>          <br />          <hr />          <div class="form-group">              <div class="col-md-offset-2 col-md-10">                  <input type="submit" value="Save" class="btn btn-default" />              </div>          </div>      </div>  }  <div>      @Html.ActionLink("Back to List", "Index")  </div> |

Création de la « View » « SelectRoleEditorViewModel».

Idem pour « SelectRoleEditorViewModel », qui est un « View » partagé que l’on place dans un sous-répertoire « EditorTemplates » de « Shared »,

Code ajouté :

|  |
| --- |
| @model Mvc5Ef6WebApiDataFirstNthW.Models.SelectRoleEditorViewModel      @Html.HiddenFor(model => model.RoleName)  <tr>      <td style="text-align:center">          @Html.CheckBoxFor(model => model.Selected)      </td>      <td>          @Html.DisplayFor(model => model.RoleName)      </td>  </tr> |

Ajout du lien d’accès à l’Action « Index » de « AccountController » dans « \_Layout »

|  |
| --- |
| <li class="divider"></li>  <li>@Html.ActionLink("Admin users", "Index", "Account")</li>  Et  <p>      @Html.ActionLink("Create New", "Register")  </p> |

Suppression du lien « Register » dans « \_LoginPartia.cshtml »

|  |
| --- |
| <ul class="nav navbar-nav navbar-right">      @\*<li>@Html.ActionLink("S’inscrire", "Register", "Account", routeValues: null, htmlAttributes: new { id = "registerLink" })</li>\*@      <li>@Html.ActionLink("Se connecter", "Login", "Account", routeValues: null, htmlAttributes: new { id = "loginLink" })</li>  </ul> |

##### Paramétrage et exécution des Migrations Entity Framework

* <http://msdn.microsoft.com/fr-fr/data/ee712907>
* <http://msdn.microsoft.com/fr-fr/data/jj591621.aspx>

Modification de « Configuration.cs »

|  |
| --- |
| internal sealed class Configuration : DbMigrationsConfiguration<Mvc5Ef6WebApiDataFirstNthW.Models.ApplicationDbContext>      {          public Configuration()          {              AutomaticMigrationsEnabled = true;          }          protected override void Seed(Mvc5Ef6WebApiDataFirstNthW.Models.ApplicationDbContext context)          {              this.AddUserAndRoles();          }          bool AddUserAndRoles()          {              bool success = false;              var idManager = new IdentityManager();              success = idManager.CreateRole("Admin");              if (!success == true) return success;              success = idManager.CreateRole("CanEdit");              if (!success == true) return success;              success = idManager.CreateRole("User");              if (!success) return success;              var newUser = new ApplicationUser()              {                  UserName = "admin",                  FirstName = "José",                  LastName = "Alvarez",                  Email = "jose.alvarez54@free.fr"              };              // Be careful here - you  will need to use a password which will              // be valid under the password rules for the application,              // or the process will abort:              success = idManager.CreateUser(newUser, "p@ssword");              if (!success) return success;              success = idManager.AddUserToRole(newUser.Id, "Admin");              if (!success) return success;              success = idManager.AddUserToRole(newUser.Id, "CanEdit");              if (!success) return success;              success = idManager.AddUserToRole(newUser.Id, "User");              if (!success) return success;              return success;          }      }  } |

|  |
| --- |
| PM> **Add-Migration Initialisation**  Structuration de la migration 'Initialisation'.  Le code du concepteur de ce fichier de migration inclut une capture instantanée de votre modèle Code First actif. Cette capture instantanée est utilisée pour calculer les modifications de votre modèle lorsque vous structurez la migration suivante. Si vous apportez des modifications supplémentaires au modèle que vous souhaitez inclure à cette migration, vous pouvez le restructurer en réexécutant 'Add-Migration Initialisation'. |



|  |
| --- |
| PM> **Update-Database**  Spécifiez l'indicateur '-Verbose' pour afficher les instructions SQL appliquées à la base de données cible.  Application de migrations explicites : [201402101226082\_Initialisation].  Application d'une migration explicite : 201402101226082\_Initialisation.  Méthode Running Seed. |

|  |  |
| --- | --- |
|  |  |
|  | |

Pour revenir en arrière

|  |
| --- |
| PM> **Update-Database -TargetMigration:201402101412074\_Initialisation -Force**  Spécifiez l'indicateur '-Verbose' pour afficher les instructions SQL appliquées à la base de données cible.  Restauration des migrations : [201402111501337\_AutomaticMigration].  Restauration d'une migration automatique : 201402111501337\_AutomaticMigration.  PM> Update-Database -Script  Aucune migration explicite en attente.  Application d'une migration automatique : 201402111526473\_AutomaticMigration. |

#### Ajout d’une photo dans le profil utilisateur

* <http://highoncoding.com/Articles/689_Uploading_and_Displaying_Files_Using_ASP_NET_MVC_Framework.aspx>
* <http://blogs.msdn.com/b/nunos/archive/2010/02/08/quick-tips-about-asp-net-mvc-editor-templates.aspx>
* <http://www.hanselman.com/blog/ASPNETMVCDisplayTemplateAndEditorTemplatesForEntityFrameworkDbGeographySpatialTypes.aspx>

La photo de l’utilisateur sera stockée dans le répertoire « **UpLoads** » situé sur la racine du site.

Le nom de l’image stockée sera : *[GUID]\_NomFichier.extension*. L’utilisateur aura le choix de ne pas avoir de photo ou de la supprimer, auquel cas il aura une image vide « BlankPhoto.jpg ».

##### Modification de la classe : « ApplicationUser »

Ajout de la propriété « PhotoUrl » qui sera stockée en base de données. La propriété Phot de type HttpPostedFileWrapper  sera décorée du DataAnnotation [NotMapped] car elle ne sert que pour l’upload du fichier.

[ <http://stackoverflow.com/questions/10385248/ignoring-a-class-property-in-entity-framework-4-1-code-first> ]

|  |
| --- |
| public class ApplicationUser : IdentityUser  {      // New properties added to extend Application User class:        [Required]      public string FirstName { get; set; }      [Required]      public string LastName { get; set; }      [Required]      public string Email { get; set; }      public string PhotoUrl { get; set; }      [NotMapped]      public HttpPostedFileWrapper Photo { get; set; }    } |

##### Modification de la classe : « RegisterViewModel »

Idem que pour « ApplicationUser.

|  |
| --- |
| public string PhotoUrl { get; set; }  [NotMapped]  public HttpPostedFileWrapper Photo { get; set; } |

Modification de la méthode « GetUser » .

|  |
| --- |
| public ApplicationUser GetUser()  {      var user = new ApplicationUser()      {          UserName = this.UserName,          FirstName = this.FirstName,          LastName = this.LastName,          Email = this.Email,          PhotoUrl = this.PhotoUrl,          Photo = this.Photo,      };      return user;  } |

##### Ajout d’une méthode static /Helpers/Utils/**SavePhotoFileToDisk**

L’enregistrement de la photo sur le disque sera réalisé par « Edit » et « Register », on consolide le code de ces méthodes en utilisant une méthode commune dont le code est donné ci-dessous.

|  |
| --- |
| /// <summary>  /// Save photo to disk  /// </summary>  /// <param name="photo">HttpPostedFileWrapper</param>  /// <param name="controller">Controller calling</param>  /// <returns>Path where photo is stored with it's calculated filename, or default photo "BlankPhoto.jpg" or null on error</returns>  public static string SavePhotoFileToDisk(HttpPostedFileWrapper photo, Mvc5Ef6WebApiDataFirstNthW.Controllers.AccountController controller, string oldPhotoUrl, bool isNoPhotoChecked)  {      string photoPath = string.Empty;      string fileName = string.Empty;        // If photo is uploaded calculate his name      if (photo != null)      {          fileName = Guid.NewGuid().ToString() + "\_" + photo.FileName;      }      else      {          // if user want to remove his photo          if (oldPhotoUrl != null && isNoPhotoChecked == true)          {              if (!oldPhotoUrl.Contains("BlankPhoto.jpg"))              {                  string fileToDelete = Path.GetFileName(oldPhotoUrl);                  var path = Path.Combine(controller.Server.MapPath("~/uploads"), fileToDelete);                  FileInfo fi = new FileInfo(path);                  if (fi.Exists)                      fi.Delete();              }          }            // If no previews photo it's a new user who don't provide photo          if (oldPhotoUrl == null || isNoPhotoChecked == true)          {              fileName = "BlankPhoto.jpg";          }          else          {              // User don't want to change his photo              return oldPhotoUrl;          }      }      // We save the new/first photo on disk      try      {          string path;          path = Path.Combine(controller.Server.MapPath("~/uploads"), fileName);          photoPath = Path.Combine(HttpRuntime.AppDomainAppVirtualPath, "uploads", fileName);          // We save the new/first photo or nothing because BlankPhoto is in the folder          if (photo != null) photo.SaveAs(path);      }      catch (Exception ex)      {          // Handled exception catch code          Helpers.Utils.SignalExceptionToElmahAndTrace(ex, controller);          return null;      }      return photoPath;  } |

L’enregistrement d’une photo n’est pas obligatoire.

« **SavePhotoFileToDisk** » offre l’utilisation de « BlankPhoto » si aucune photo n’est fournie lors de l’enregistrement, ainsi que la suppression de la photo existante.

##### Modification de « Register » de : « AccountController »

|  |
| --- |
| [HttpPost]  [Authorize(Roles = "Admin")]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Register(RegisterViewModel model)  {      if (ModelState.IsValid)      {          // Save file to disk and retreive calculated file name or null if handled exception occure          // if user don't provide photo then he don't want photo          model.PhotoUrl = Utils.SavePhotoFileToDisk(model.Photo, this, null, model.Photo == null ? true:false);            var user = model.GetUser();            var result = await UserManager.CreateAsync(user, model.Password);            if (result.Succeeded)          {              var idManager = new IdentityManager();              idManager.AddUserToRole(user.Id, "User");              return RedirectToAction("Index", "Account");          }          else          {              AddErrors(result);          }      }        // Si nous sommes arrivés là, un échec s’est produit. Réafficher le formulaire      return View(model);  } |

##### Modification de : « /Views/Acount/Register.cshtml »

|  |
| --- |
| @using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizontal", role = "form" , enctype ="multipart/form-data" })) |

|  |
| --- |
| <div class="form-group">      @Html.LabelFor(m => m.Photo, new { @class = "col-md-2 control-label" })      <div class="col-md-10">          @Html.TextBoxFor(m => m.Photo, null, new { type = "file", onchange = "readURL(this)", title = "Search for file to add", @class = "input-file  col-md-offset-1" })          @Html.ValidationMessageFor(m => m.Photo)<br />          <img id="blah" src="#" alt="" width="100" height="100" />          <img src="@Html.DisplayFor(model => model.PhotoUrl)" alt="@Html.DisplayFor(model => model.UserName) " title="@Html.DisplayFor(model => model.FirstName)" />      </div>  </div> |

##### Modification de : « /Views/Acount/Index.cshtml »

|  |
| --- |
| <th>      @Html.DisplayNameFor(model => model.PhotoUrl)  </th> |

|  |
| --- |
| <td>      <img src="@item.PhotoUrl" alt="@item.UserName" title="@(item.FirstName + ' ' + item.LastName)" />  </td> |

##### Modification de la classe : « EditUserViewModel ».

Pour afficher la photo dans « **Edit.cshtml** » et « **Delete.cshtml** », ces deux « View » utilisant le même modèle « **EditUserViewModel** ».

On ajoute la même chose que pour « **Register.cshtml** », avec en plus la propriété :

|  |
| --- |
| [NotMapped]  [Display(Name = "Remove if checked")]  public bool IsNoPhotoChecked { get; set; } |

Pour gérer le checkbox de suppression d’une photo.

##### Modification de la méthode « Edit » de : « AccountController »

|  |
| --- |
| [HttpPost]  [Authorize]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Edit(EditUserViewModel model)  {      var Db = new ApplicationDbContext();      var user = Db.Users.First(u => u.UserName == model.UserName);      if (ModelState.IsValid)      {          model.PhotoUrl = Utils.SavePhotoFileToDisk(model.Photo, this, user.PhotoUrl, model.IsNoPhotoChecked);          user.PhotoUrl = model.PhotoUrl;            // Update the user data:          user.FirstName = model.FirstName;          user.LastName = model.LastName;          user.Email = model.Email;          Db.Entry(user).State = System.Data.Entity.EntityState.Modified;          await Db.SaveChangesAsync();            if (HttpContext.User.IsInRole("Admin"))          {              return RedirectToAction("Index");          }          else          {              return RedirectToAction("Edit", new { id = user.UserName });          }      }      model.PhotoUrl = user.PhotoUrl;      // If we got this far, something failed, redisplay form      return View(model);  } |

##### Modification de « DeleteConfirmed» de : « AccountController »

|  |
| --- |
| public ActionResult DeleteConfirmed(string id)  {      if (id != "admin")      {          var Db = new ApplicationDbContext();          var user = Db.Users.First(u => u.UserName == id);          // Dont delete BlankPhoto.jpg          if (!user.PhotoUrl.Contains("BlankPhoto.jpg"))          {              var path = Server.MapPath("~") + user.PhotoUrl;              FileInfo fi = new FileInfo(path);              if (fi.Exists)                  fi.Delete();          }          Db.Users.Remove(user);          Db.SaveChanges();      }      return RedirectToAction("Index");  } |

NB : On prend soin de ne pas supprimer ni l’utilisateur « admin », ni « BlankPhoto ».

Suppression du fichier photo lors de la suppression d’un utilisateur

##### Modification de : « /Views/Account/Edit.cshtml »

|  |
| --- |
| @using (Html.BeginForm("Edit", "Account", FormMethod.Post, new { enctype ="multipart/form-data" })) |

|  |
| --- |
| <div class="form-group">      @Html.LabelFor(m => m.Photo, new { @class = "col-md-2 control-label" })      <div class="col-md-10">          @Html.TextBoxFor(m => m.Photo, null, new { id = "FilePhoto", onchange = "readURL(this)", type = "file", title = "Search for file to add", @class = "input-file btn-warning" })          @Html.ValidationMessageFor(m => m.Photo)<br />          <img id="blah" src="#" alt="" width="100" height="100" />      </div>  </div>    <div class="form-group">      @Html.Label("Actual photo", new { @class = "col-md-2 control-label" })      <div class="col-md-10">          <img src="@Html.DisplayFor(model => model.PhotoUrl)" alt="@Html.DisplayFor(model => model.UserName) " title="@Html.DisplayFor(model => model.FirstName)" />      </div>  </div>    @\* if user don't have photo don't propose to remove it\*@  @if (!(Model.PhotoUrl.Contains("BlankPhoto.jpg")))  {      <div class="form-group">          @Html.Label("Your choice", new { @class = "col-md-2 control-label" })          <div class="col-md-10">              @Html.CheckBox("NoPhoto", false, new { id = "NoPhoto" }) Clear photo if checked          </div>      </div>  } |

##### Modification de : « /Views/Account/Delete.cshtml »

|  |
| --- |
| <dt>      @Html.DisplayNameFor(model => model.PhotoUrl)  </dt>    <dd>      <img src="@Html.DisplayFor(model => model.PhotoUrl)" alt="@Html.DisplayFor(model => model.UserName) " title="@Html.DisplayFor(model => model.FirstName)" />  </dd> |

##### Mise à jour de la base de données :

Ajout de la Migration « Photo »

|  |
| --- |
| PM> **Add-Migration Photo**  Structuration de la migration 'Photo'.  Le code du concepteur de ce fichier de migration inclut une capture instantanée de votre modèle Code First actif. Cette capture instantanée est utilisée pour calculer les modifications de votre modèle lorsque vous structurez la migration suivante. Si vous apportez des modifications supplémentaires au modèle que vous souhaitez inclure à cette migration, vous pouvez le restructurer en réexécutant 'Add-Migration Photo'. |

Cela a pour effet l’ajout du fichier « 201402120824406\_Photo»

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW.Migrations  {      using System;      using System.Data.Entity.Migrations;        public partial class Photo : DbMigration      {          public override void Up()          {              AddColumn("dbo.AspNetUsers", "PhotoUrl", c => c.String());          }            public override void Down()          {              DropColumn("dbo.AspNetUsers", "PhotoUrl");          }      }  } |

Pour mettre à jour la base de données

|  |
| --- |
| PM> **Update-Database -TargetMigration:** **201402120824406\_Photo**  Spécifiez l'indicateur '-Verbose' pour afficher les instructions SQL appliquées à la base de données cible.  Application de migrations explicites : [201402111528351\_Photo].  Application d'une migration explicite : 201402111528351\_Photo.  Méthode Running Seed. |

##### Ajout modification du profil par un utilisateur

Pour l’instant, seul l’administrateur peut modifier le profil des utilisateurs, ces derniers ne peuvent que changer leur mot de passe. Dans cette partie, nous allons offrir à l’utilisateur la possibilité de modifier l’intégralité de son profil.

###### Modification des « Views »

Manage.cshtml

Modification du titre, de "Gérer le compte" en "Changer le mot de passe"

\_LoginPartial.cshtml

Modification du lien Action, de « Manage » en « Edit » et de routeValues: de null en new{id=User.Identity.GetUserName()}

|  |
| --- |
| @Html.ActionLink("Bonjour " + User.Identity.GetUserName() + "!", "Edit", "Account", routeValues: new{id=User.Identity.GetUserName()}, htmlAttributes: new { title = "Administrer"}) |

###### Modification de AccountController.cs

Modification du DataAnnotation de la méthode « Edit », de [Authorize(Roles = "Admin")] en [Authorize]

|  |
| --- |
| [Authorize]  public ActionResult Edit(string id, ManageMessageId? Message = null) |

|  |
| --- |
| [HttpPost]  [Authorize]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Edit(EditUserViewModel model) |

Modification de la redirection :

|  |
| --- |
| [HttpPost]  [Authorize]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Edit(EditUserViewModel model)  {  ………………………………………………  if(HttpContext.User.IsInRole("Admin"))  {      return RedirectToAction("Index");  }  else  {      return RedirectToAction("Edit", new { id = user.UserName });  }  ………………………………………………  } |

##### Ajout DataAnnotation : [ValidateFile]

Ajout de la classe : « ValidateFileAttribute : ValidationAttribute »

|  |
| --- |
| /// <summary>  /// Customized data annotation validator for uploading file  /// http://www.dotnet-tricks.com/Tutorial/mvc/aX9D090113-File-upload-with-strongly-typed-view-and-model-validation.html  /// </summary>  public class ValidateFileAttribute : ValidationAttribute  {      public override bool IsValid(object value)      {          string uri = HttpContext.Current.Request.Url.ToString();          int maxContentLength = 512 \* 512 \* 3; // photo color = 3 x 8 bits = 3 bytes          int maxHeight = 200;          int maxWidth = 150;          string[] AllowedFileExtensions = new string[] { ".jpg", ".gif", ".png"};          var file = value as HttpPostedFileBase;          // user don't want photo is permitted          if (file == null)              return true;              // test photo type          else if (!AllowedFileExtensions.Contains(file.FileName.Substring(file.FileName.LastIndexOf('.'))))          {              ErrorMessage = string.Format( "Please upload Your Photo of type: {0}" , string.Join(", ", AllowedFileExtensions));              return false;          }              // test photo size          else if (file.ContentLength > maxContentLength)          {              ErrorMessage = string.Format("Your Photo is too large, maximum allowed size is : {0} MB" , (maxContentLength / 1024).ToString());              return false;          }          else          {              // test photo dimensions              using (System.Drawing.Image myImage = System.Drawing.Image.FromStream( file.InputStream))              {                  if (myImage.Height > maxHeight && myImage.Width > maxWidth)                  {                      ErrorMessage = string.Format("Your Photo is too large, maximum allowed size is : Width {0} x Height {1} pixels", maxWidth, maxHeight);                      return false;                  }              }          }              return true;      }  } |

Ajout de l’Annotation aux propriétés

Décoration avec [**ValidateFile**] de la propriété **Photo** de type **HttpPostedFileWrapper** dans « **EditUserViewModel** » et « **RegisterUserViewModel** »

Modification du « Controller »

Simplication de la méthode **Register(RegisterViewModel model)**, le travail étant fait dans la classe « ValidateFileAttribute »

Modification des « Views »

Modification des <div class="form-group"> dans Edit.cshtml and Register.cshtml, pour l’affichage de l’erreur et remplacement de <input type="file" name="photo" /> par @Html.TextBoxFor(m => m.Photo, null, new { type = "file", title = "Search for file to add", @class = "input-file" })

|  |
| --- |
| <div class="form-group">      @Html.LabelFor(m => m.Photo, new { @class = "col-md-2 control-label" })      <div class="col-md-10">          @Html.TextBoxFor(m => m.Photo, null, new { type = "file", title = "Search for file to add", @class = "input-file" })          @Html.ValidationMessageFor(m => m.Photo)          <img src="@Html.DisplayFor(model => model.PhotoUrl)" alt="@Html.DisplayFor(model => model.UserName) " title="@Html.DisplayFor(model => model.FirstName)" />      </div>  </div> |

### Design & Razor & BootStrap & Rotativa

Tous les modèles de projet de Visual Studio 2013 utilisent la librairie « JQUERY », et sont construits avec Bootstrap 3.

Bootstrap 3 est un framework CSS de type « Responsive » qui est utlisé pour créer des sites web. Bootstrap est livré dans l’application à l'aide de la fonctionnalité d'optimisation Web (Web Optimization). C’est dans App\_Start\BundleConfig.cs que les fichiers Bootstrap sont regroupés. Cf. lien <http://www.asp.net/mvc/tutorials/mvc-4/bundling-and-minification> qui donne les avantages du « Bundling » et de la « Minification ».

Les templates peuvent utiliser les moteurs de rendu html (Aspx ou Razor) Razor a été conçu pour faciliter la conception des pages ASP.NET. Il introduit une syntaxe de programmation assez facilement compréhensible, qui vous permet d'insérer du code serveur dans une page Web qui peut également contenir du HTML, du CSS et des scripts JavaScript.

Le moteur Razor est fluide, compact, expressif et s'appuie sur la syntaxe des langages .NET C# et Visual Basic .NET. Il offre la puissance d'ASP.NET pour la création rapide des applications Web fonctionnelles et sophistiquées.

#### Viewport

Dans le MasterPage « \_Layout », la balise meta :

|  |
| --- |
| <meta name="viewport" content="width=device-width, initial-scale=1.0"> |

Permet de contrôler la mise en page sur les navigateurs mobiles.

La propriété width contrôle la taille du viewport. Elle peut être réglée à une valeur précise de pixels, comme width=600, ou bien à la valeur spéciale device-width qui correspond à la largeur de l'écran en pixels CSS à l'échelle 100%. (Il existe les propriétés et valeurs height et device-height correspondantes, qui peuvent être utiles pour les pages contenant des éléments qui changent de taille ou de position en fonction de la hauteur du viewport.)

La propriété initial-scale contrôle le niveau de zoom lorsque la page est chargée pour la première fois. Les propriétés maximum-scale, minimum-scale, et user-scalable contrôlent la manière dont les utilisateurs sont autorisés à zoomer ou dézoomer une page.

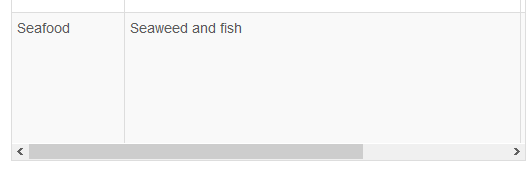
#### Table Category

Deux solutions sont offertes :

Table Category-responsive

Créer des tableaux « Responsive » en wrappant les.   « .table » en. « .table-responsive », permettant de faire défiler les tableaux horizontalement pour de petits appareils (sous 768px).

On utilise aussi les classes : table table-striped table-bordered table-hover table-condensed.



Cf. <http://getbootstrap.com/css/#tables>

|  |
| --- |
| <div class="table-responsive">      <table class="table table-striped table-bordered table-hover table-condensed">          <tr>              <th>                  @Html.DisplayNameFor(model => model.CategoryName)              </th>  ……………………………………………………………….  @foreach (var item in Model)  {      <tr>          <td data-title=@Html.DisplayNameFor(model => model.CategoryName)>              @Html.DisplayFor(modelItem => item.CategoryName)          </td>          <td data-title=@Html.DisplayNameFor(model => model.Description)>              @Html.DisplayFor(modelItem => item.Description)          </td>  ………………………………………………………………. |

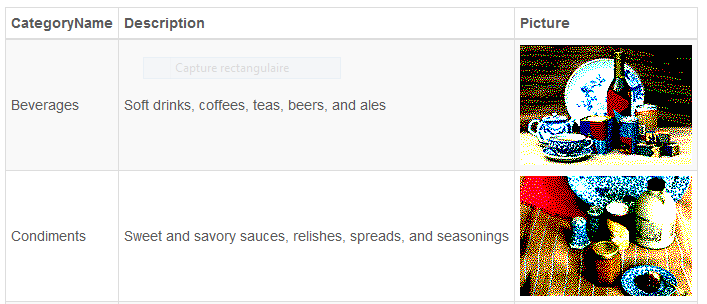
On utilise ici les méthodes d’extension « **@Html.DisplayNameFor** » et « **@Html.DisplayFor** ».

* **DisplayNameFor** : Retourne le « Display name » du model.
* **DisplayFor** : Retourne une chaîne qui contient chaque valeur de propriété de l'objet représenté par l'expression Expression, à l'aide du modèle spécifié.

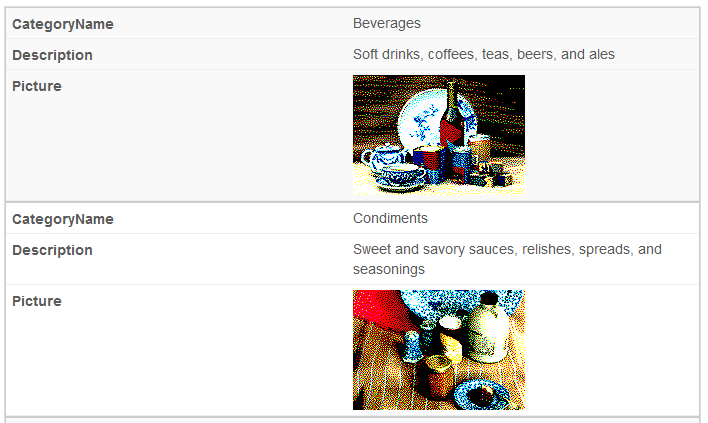
Table Category « no more table »

Utilise la balise HTML 5 « section » (Cf. <http://www.alsacreations.com/article/lire/1376-html5-section-article-nav-header-footer-aside.html> )

|  |
| --- |
| <section class="container-fluid" id="no-more-tables" >      <table class="table-bordered table-striped table-condensed cf">            <thead class="cf"> |



Transformée en :



#### PDF : Rotativa

Cf. :

* <http://letsfollowtheyellowbrickroad.blogspot.it/search/label/Rotativa>
* <https://github.com/webgio/Rotativa>
* <https://www.nuget.org/packages/Rotativa/>

Permet de créer des PDF.

Deux Views utilisent cet librairie :

* PrintIndexAsPdf.cshtml
* PrintIndexAsPdfPageList.cshtml

On utilise aussi une feuille de style et @media pour cacher les parties ne devant pas apparaitre à l’impression ou dans les PDF :

|  |
| --- |
| @section styles      {      <style type="text/css">          @@media print {              footer, #tdNotPrinted, #divNotPrinted, #pNotPrinted {                  display: none;              }          }            @@page {              margin: 0cm;          }      </style>  } |

|  |
| --- |
| <p id="pNotPrinted">      @Html.ActionLink("Create New", "Create")  </p> |

#### Googlemaps dans contact.cshtml

Web.Config :

|  |
| --- |
| <add key="cdf54.CompanyAddress" value="1 rue Jeannot Nancy France" />      <add key="cdf54.GoogleMapKey" value="AIzaSyALDo0MgNC8vRSNN3It38Yo9L\_g8\_8GoqM" /> |

Helpers/Utils

|  |
| --- |
| public static string GetConfigCompanyAddress()  {      return ConfigurationManager.AppSettings["cdf54.CompanyAddress"].ToString();  }  public static string GetGoogleMapKey()  {      return ConfigurationManager.AppSettings["cdf54.GoogleMapKey"].ToString();  } |

Scripts

|  |
| --- |
| @section script\_google  {      <script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false&key=@Utils.GetGoogleMapKey()"></script>  } |

|  |
| --- |
| @section scripts{      <script type="text/javascript">          $(document).ready(function () {              var geocoder = new google.maps.Geocoder();              var infowindow = new google.maps.InfoWindow();              var latlng = new google.maps.LatLng(-34.397, 150.644);              var mapOptions = {                  zoom: 15,                  center: latlng              }              var map = new google.maps.Map(document.getElementById("map-canvas"), mapOptions);              var address = $('address').text();                geocoder.geocode({ 'address': address }, function (results, status) {                  if (status == google.maps.GeocoderStatus.OK) {                      map.setCenter(results[0].geometry.location);                      var marker = new google.maps.Marker({                          map: map,                          position: results[0].geometry.location                      });                      infowindow.setContent(results[0].formatted\_address);                      infowindow.open(map, marker);                  } else {                      alert("Geocode was not successful for the following reason: " + status);                  }              });                //Convert address tags to google map links - Copyright Michael Jasper 2011              $('address').each(function () {                  var link = "<a href='http://maps.google.com/maps?q=" + encodeURIComponent($(this).text()) + "' target='\_blank'>" + $(this).text() + "</a>";                  $(this).html(link);              });          });      </script>  } |

#### Razor HtmlHelper déclaratif ou méthode d’extension de HtmlHelper

Affichage des images de la table « Category »

Réf :

* <http://rdonfack.developpez.com/tutoriels/dotnet/creation-pages-web-asp-net-utilisant-moteur-vue-razor/?page=page_6>

Dans la « View » « Index » correspondant au « Controller » « Category » le tableau doit afficher des images extraites de la base de données.

Les images sont retournées par le « Model » en un type « byte[] ». Razor doit générer sur cette page, une balise HTML image « img » :

|  |
| --- |
| <img class="xxx" src="xxx" title="xxx" alt="xxx" border="xxx" /> |

Il s’agit donc dans la boucle:

@foreach (var item in Model)

qui affiche les lignes du tableau, de convertir le tableau de bytes en une chaîne de caractères « src ».

Le code pour effectuer cette conversion est le suivant :

|  |
| --- |
| byte[] photo = picture;  string imageSrc = null;  if (photo != null)  {      MemoryStream ms = new MemoryStream();      ms.Write(photo, 78, photo.Length - 78); // strip out 78 byte OLE header (don't need to do this for normal images)      string imageBase64 = Convert.ToBase64String(ms.ToArray());      imageSrc = string.Format("data:image/png;base64,{0}", imageBase64);  } |

*Où « picture » est le tableau de bytes à convertir.*

Réf : <http://www.nullskull.com/a/10450951/aspnet-mvc-display-images-directly-from-the-viewmodel-into-your-views.aspx>

On peut l’implémenter de plusieurs manières :

|  |
| --- |
| **En insérant le code dans la vue Index :**  @{byte[] photo = item.Picture;      string imageSrc = null;      if (photo != null)      {          MemoryStream ms = new MemoryStream();          ms.Write(photo, 78, photo.Length - 78);          string imageBase64 = Convert.ToBase64String(ms.ToArray());          imageSrc = string.Format("data:image/png;base64,{0}", imageBase64);      }  }  <td data-title=@Html.DisplayNameFor(model => model.Picture)>  <img class="img-responsive" src="@imageSrc" title="@item.CategoryName" alt="@item.CategoryName" border="0" /> |
| **En déporttant le code dans une méthode static:**  <img class="img-responsive" src="@Utils.ByteToStringImage(item.Picture)" title="@item.CategoryName " alt="@item.CategoryName " border="0" /> |
| public class Utils  {      public static String ByteToStringImage(byte[] picture)      {          byte[] photo = picture;          string imageSrc = null;          if (photo != null)          {              MemoryStream ms = new MemoryStream();              ms.Write(photo, 78, photo.Length - 78); // strip out 78 byte OLE header (don't need to do this for normal images)              string imageBase64 = Convert.ToBase64String(ms.ToArray());              imageSrc = string.Format("data:image/png;base64,{0}", imageBase64);          }          return imageSrc;      }  } |
| **En utilisant un HtmlHelper “Image” et la méthode statique précédente:**  @using Mvc5Ef6WebApiDataFirstNthW.Helpers  @Html.Image("img-responsive", Utils.ByteToStringImage(item.Picture), item.CategoryName, item.CategoryName, "0") |
| public static class MyHtmlHelpers  {      public static MvcHtmlString Image(this HtmlHelper helper, string imgClass, string imgSrc, string imgTitle, string imgAlt, string imgBorder)      {          var builder = new TagBuilder("img");          builder.MergeAttribute("class", imgClass);          builder.MergeAttribute("src", imgSrc);          builder.MergeAttribute("title", imgTitle);          builder.MergeAttribute("alt", imgAlt);          builder.MergeAttribute("border", imgBorder);          return MvcHtmlString.Create(builder.ToString(TagRenderMode.SelfClosing));      }  } |
| **En utilisant un helper Razor TagImage, faisant lui-même appel à la méthode statique de conversion :**  @MyRazorHelpers.TagImage("img-responsive", item.Picture, @item.CategoryName, @item.CategoryName, "0") |
| @using Mvc5Ef6WebApiDataFirstNthW.Helpers  @helper TagImage(string imgClass, byte[] picture, string imgTitle, string imgAlt, string imgBorder)  {      <img class=@imgClass src=@Utils.ByteToStringImage(picture) title=@imgTitle alt=@imgAlt border=@imgBorder />  } |

Cette dernière étant la plus judicieuse, puisque le code razor dans la vue est minimal, et la méthode Utils.ByteToStringImage peut être réutilisée pour d’autres circonstances.

### Area « Administration »

Les « Areas » permettent d’organiser le code.

Ici, ils sont utilisés pour séparer la partie applicative qui implémente ses « View/Controller/Model » à la racine du site du trio pour la partie Administration.

L’Area Administration implémente un Controller « Home » et une View « Index ».

La prise en compte des routes sans utilisation des Areas est définie dans App\_Start/RouteConfig.cs

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW  {      public class RouteConfig      {          public static void RegisterRoutes(RouteCollection routes)          {              routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");              routes.MapRoute(                  name: "Default",                  url: "{controller}/{action}/{id}",                  defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }              );          }      }  } |

Dans la mesure où l’Area « Administration » utilise elle aussi un Controller « Home », il faut différentier la route par défaut : <http://URL_SITE> qui est <http://URL_SITE/Home/index> de la route de l’Area définie dans Areas/Administration/AdministrationAreaRegistration.cs

|  |
| --- |
| public class AdministrationAreaRegistration : AreaRegistration      {          public override string AreaName          {              get              {                  return "Administration";              }          }            public override void RegisterArea(AreaRegistrationContext context)          {              context.MapRoute(                  "Administration\_default",                  "Administration/{controller}/{action}/{id}",                  new { controller="Home", action = "Index", id = UrlParameter.Optional }              );          }      }  } |

Si on ne veut pas avoir une erreur du genre :

|  |
| --- |
| Server Error in '/' Application.  Multiple types were found that match the controller named 'Home'. This can happen if the route that services this request ('{controller}/{action}/{id}') does not specify namespaces to search for a controller that matches the request. If this is the case, register this route by calling an overload of the 'MapRoute' method that takes a 'namespaces' parameter.  The request for 'Home' has found the following matching controllers:  Mvc5Ef6WebApiDataFirstNthW.Areas.Administration.Controllers.HomeController  Mvc5Ef6WebApiDataFirstNthW.Controllers.HomeController |

Pour ce faire, on rajoute le namespace du contrôleur situé sur la racine soit :

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW  {      public class RouteConfig      {          public static void RegisterRoutes(RouteCollection routes)          {              routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");              routes.MapRoute(                  name: "Default",                  url: "{controller}/{action}/{id}",                  defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional },                  namespaces: new[] {"Mvc5Ef6WebApiDataFirstNthW.Controllers"}              );          }      }  } |

Utilisation :

<http://URL_SITE/Administration> ou   
<http://URL_SITE/Administration/Home> ou  
<http://URL_SITE/Administration/Home/Index>

Et dans un lien en Razor :

|  |
| --- |
| @Html.ActionLink("Protected page", "Index", "Home", new { area = "Administration"},null ) |

#### Areas/Administration/Controllers/Home

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW.Areas.Administration.Controllers  {      public class HomeController : Controller      {          //          // GET: /Administration/Home/          [Authorize(Users="toto")]          public ActionResult Index()          {              return View();          }      }  } |

Annotation : [Authorize(Users="toto")] seule l’utilisateur authentifié « toto », peut accéder à cette page.

#### Areas/Administration/Views/Index.cshtml

Elle fait appel à un helper Razor : « AdministrationInfo » dont le « MyRazorHelper » est situé dans App\_Code pour qu’il puisse être vu dans toute l’application, puisqu’il implémente d’autres helper Razor.

|  |
| --- |
| @{      ViewBag.Title = "Main Administration";  }  <h2>@ViewBag.Title</h2>    @MyRazorHelpers.AdminInformations() |

« AdminInformation » utilise : les espaces de noms suivants:

* System.Reflection.Assembly,
* System.Configuration.ConfigurationManager,
* System.Web.HttpContext,
* System.Threading.Thread.CurrentThread

Pour afficher un certain nombre d’informations techniques.

Cf. code dans App\_Code/MyRazorHelpers.cshtml

#### Areas/Administration/Views/Shared/\_Layout

Lien de retour vers l’application principale :

|  |
| --- |
| <div class="navbar-collapse collapse">      <ul class="nav navbar-nav">          <li>@Html.ActionLink("MainApp", "Index", "Home", new { area = "" }, null)</li>      </ul>  </div> |

Il nous faut spécifier : new { area = "" } car le site racine et l’Area « Administration » utilisent le même nom de « Controller » (« Home ».

### WebApi

Le support d'ASP.NET Web API dans le framework ASP.NET MVC permet de créer facilement des services Web RestFull qui peuvent être consommés à partir d'un large éventail de clients (réseaux sociaux, navigateurs, terminaux mobiles, etc.).

Une application MVC diffère d’une WebApi en deux points :

* Le « **controller** » d’une application MVC dérive de la classe « **Controller** », alors que le « **controller** » d’un WebApi dérive de la classe « **ApiController** »
* Le routage est aussi différent :

|  |  |
| --- | --- |
| Application MVC | "{controller}/{action}/{id}" |
| WebApi | "api/{controller}/{id}" |

Le WebApi n’a pas besoin de l’action, il déduit l’action à effectuer à partir de la requête http.

Exemples :

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode HTTP | Chemin URI | Action | Description |
| Get | GET api/Category | GetCategories() | Retourne toutes les données comme un IEnumerable. |
| Get | GET api/Product/5 | GetProductById(int id) | Retourne l'élément ayant pour Id 5. |
| Post | POST api/Product | PostProduct(Product product) | Enregistrement d'un nouvel élément. |
| Put | PUT api/Product/5 | PutProduct(int id, Product product) | Modification de l'élément ayant pour Id 5. |
| Delete | DELETE api/Product/5 | DeleteProduct(int id) | Surpression de l'élément ayant pour Id 5. |

Par convention, le nom de l’action doit commencer par Get, Post, Put ou Delete

#### App\_Start/ WebApiConfig.cs

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW  {      public static class WebApiConfig      {          public static void Register(HttpConfiguration config)          {              //// Web API configuration and services              //// http://www.asp.net/web-api/overview/formats-and-model-binding/json-and-xml-serialization              //config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();              //config.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;              // Web API routes              config.MapHttpAttributeRoutes();                config.Routes.MapHttpRoute(                  name: "DefaultApi",                  routeTemplate: "api/{controller}/{id}",                  defaults: new { id = RouteParameter.Optional }              );          }      }  } |

#### Les « Controllers » implémentés sont dans le répertoire « WebApi ».

|  |
| --- |
| public class CategoryController : ApiController  public IQueryable<Category> GetCategories()  public Category GetCategoryById(int id)  public IEnumerable<Category> GetCategoryByName(string name)  public class ProductController : ApiController  public IQueryable<Product> GetProducts()  public async Task<IHttpActionResult> GetProduct(int id)  public async Task<IHttpActionResult> PutProduct(int id, Product product) |

#### Clients WebApi

##### Clients web html

|  |
| --- |
| <head>      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />      <title></title>      <link href="/NorthWind/Content/cdf54.bootstrap.css" rel="stylesheet" />      <link href="/NorthWind/Content/bootstrap.css" rel="stylesheet" />      <script src="/NorthWind/Scripts/jquery-2.0.3.js" type="text/javascript"></script>        <script type="text/javascript">          $(document).ready(function () {              //http://msdn.microsoft.com/fr-fr/magazine/dn532203.aspx              //http://www.asp.net/web-api/overview/security/enabling-cross-origin-requests-in-web-api              var webApiUri = 'http://' + window.location.host + '/NorthWind/API/Category';              var $categories = $("#categoriesection");              $.ajax({                  type: "GET",                  contentType: "application/json; charset=utf-8",                  url: webApiUri,                  dataType: "json",                  error: function (request, status, error) {                      alert(request.responseText);                  },                  success: function (data) {                      //console.debug(data)                      $('#home').attr('href', 'http://' + window.location.host + '/NorthWind');                      $.each(data, function (key, val) {                          $categories.append("<tr><td>" + val.categoryID + "</td><td>" + val.categoryName + "</td><td>" + val.description + "</td></tr>");                      });                  }              })          });      </script>    </head> |
| <table class="table table-striped table-bordered table-hover table-condensed">      <thead style=" background-color: #006DCC; color: white;">          <tr>              <th>CategoryID</th>              <th>CategoryName</th>              <th>Description</th>          </tr>      </thead>      <tbody id="categoriesection"></tbody>  </table> |

Avec quelques adaptations, idem pour « Products » en remplaçant « Category » par « Product ».

##### Clients web MVC avec JQuery

« Controller »

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW.Controllers  {      public class WebApiClientController : Controller      {          /// <summary>          /// GET: /WebApiClientViewCategoriesJQuery/          /// The view retreive data using JQuery/Ajax          /// </summary>          /// <returns></returns>          public ActionResult WebApiClientViewCategoriesJQuery()          {              return View();          }      }  } |

« View »

|  |
| --- |
| @section scripts  {  <script type="text/javascript">      $(document).ready(function () {          var $products = $("#categoriesection");          $.ajax({              type: "GET",              url: '@(new Uri("http://" + Request.Url.Authority + "/NorthWind/").ToString() + "API/category")',              contentType: "application/json; charset=utf-8",              dataType: "json",              error: function (request, status, error) {                  alert(request.responseText);              },              success: function (data) {                  //console.debug(data)                  $.each(data, function (key, val) {                      $products.append("<tr><td>" + val.categoryID + "</td><td>" + val.categoryName + "</td><td>" + val.description + "</td></tr>");                    });              }          })      });  </script>  }  <table id="sortTableExample" class="table table-striped table-bordered table-hover table-condensed">      <thead style=" background-color: #006DCC; color: white;">          <tr>              <th>CategoryID</th>              <th>CategoryName</th>              <th>Description</th>          </tr>      </thead>      <tbody id="categoriesection"></tbody>  </table> |

Avec quelques adaptations, idem pour « Products » en remplaçant « Category » par « Product ».

##### Clients web MVC avec accès à WebApi depuis le controller »

« Controller »

|  |
| --- |
| /// <summary>  /// GET: /WebApiClientViewCategoriess/  /// C# code in the controller to retreive data from WebApi server.  /// </summary>  /// <returns></returns>  public ActionResult WebApiClientViewCategories()  {      IEnumerable<Mvc5Ef6WebApiDataFirstNthW.Models.CategoriesWebApiViewModel.Category> categories = null;      HttpClient client = new HttpClient();      client.BaseAddress = new Uri("http://" + Request.Url.Authority + "/NorthWind/");        // Add an Accept header for JSON format.      client.DefaultRequestHeaders.Accept.Add(          new MediaTypeWithQualityHeaderValue("application/json"));      // List all products.      HttpResponseMessage response = client.GetAsync("api/Category").Result;  // Blocking call!      if (response.IsSuccessStatusCode)      {          try          {              categories = response.Content.ReadAsAsync<IEnumerable<Mvc5Ef6WebApiDataFirstNthW.Models.CategoriesWebApiViewModel.Category>>().Result;          }          catch (Exception ex)          {              Helpers.Utils.SignalExceptionToElmahAndTrace(ex, this);              return RedirectToAction("HandledFatalException", "MyError", new { area = "" });          }      }      return View(categories);  } |

Utilise la classe « **HttpClient** » pour envoyer des requêtes HTTP et recevoir des réponses HTTP d'une ressource identifiée par un URI.

« ViewModel »

|  |
| --- |
| namespace Mvc5Ef6WebApiDataFirstNthW.Models  {      public class CategoriesWebApiViewModel      {          public partial class Category          {              public int CategoryID { get; set; }              public string CategoryName { get; set; }              public string Description { get; set; }              public byte[] Picture { get; set; }          }      }  } |

« View »

|  |
| --- |
| @model IEnumerable<Mvc5Ef6WebApiDataFirstNthW.Models.CategoriesWebApiViewModel.Category>  @{      ViewBag.Title = "Categories";  }  <h2>WebApi Client Categories with MVC View and data retreive in controller </h2>    <table class="table">      <tr>          <th>              @Html.DisplayNameFor(model => model.CategoryName)          </th>          <th>              @Html.DisplayNameFor(model => model.Description)          </th>          <th>              @Html.DisplayNameFor(model => model.Picture)          </th>          <th></th>      </tr>    @foreach (var item in Model) {      <tr>          <td>              @Html.DisplayFor(modelItem => item.CategoryName)          </td>          <td>              @Html.DisplayFor(modelItem => item.Description)          </td>          <td>              @MyRazorHelpers.TagImage("img-responsive", item.Picture, @item.CategoryName, @item.CategoryName, "0")          </td>      </tr>  }    </table> |

Avec quelques adaptations, idem pour « Products » en remplaçant « Category » par « Product ».

##### Liens dans \_Layout

|  |
| --- |
| <li class="nav-header">WebApi</li>  <li>@Html.ActionLink(\_Layout.ProductsAPI, "Product", "API", null, new { target = "\_blank" })</li>  <li>@Html.ActionLink(\_Layout.CategoriesAPI, "Category", "API", null, new { target = "\_blank" })</li>  <li>@Html.ActionLink(\_Layout.ValuesAPI, "Values", "API", null, new { target = "\_blank" })</li>  <li>@Html.ActionLink(\_Layout.VersionAPI, "Version", "API", null, new { target = "\_blank" })</li>  <li class="divider"></li>  <li class="nav-header">WebApi clients</li>  <li>@Html.ActionLink(\_Layout.ViewProductsJQuery, "WebApiClientViewProductsJQuery", "WebApiClient")</li>  <li>@Html.ActionLink(\_Layout.ViewProducts, "WebApiClientViewProducts", "WebApiClient")</li>  <li><a href="~/StaticsWebApiClients/WebApiProducts.html">@\_Layout.HtmlProductsJQuery</a></li>  <li>@Html.ActionLink(\_Layout.ViewCategoriesJQuery, "WebApiClientViewCategoriesJQuery", "WebApiClient")</li>  <li>@Html.ActionLink(\_Layout.ViewCategories, "WebApiClientViewCategories", "WebApiClient")</li>  <li><a href="~/StaticsWebApiClients/WebApiCategories.html">@\_Layout.HtmlCategoriesJQuery</a></li> |

##### Client Console

|  |
| --- |
| public class Product          {              public int ProductID { get; set; }              public string ProductName { get; set; }              public int SupplierID { get; set; }              public int CategoryID { get; set; }              public string QuantityPerUnit { get; set; }              public decimal UnitPrice { get; set; }              public Int16 UnitsInStock { get; set; }              public Int16 UnitsOnOrder { get; set; }              public bool Discontinued { get; set; }              public Int16 ReorderLevel { get; set; }          }          static void Main(string[] args)          {              IEnumerable<Product> products = GetProducts();              PrintUnitsInStock(products);              Console.ReadLine();              PrintProducts(products);              Console.ReadLine();          }          private static IEnumerable<Product> GetProducts()          {              IEnumerable<Product> products = null;              HttpClient client = new HttpClient();              client.BaseAddress = new Uri(ConfigurationManager.AppSettings["adressWebApiServer"].ToString());              // Add an Accept header for JSON format.              client.DefaultRequestHeaders.Accept.Add(                  new MediaTypeWithQualityHeaderValue("application/json"));              // List all products.              HttpResponseMessage response = client.GetAsync("api/Product").Result;  // Blocking call!              if (response.IsSuccessStatusCode)              {                  try                  {                      products = response.Content.ReadAsAsync<IEnumerable<Product>>().Result;                  }                  catch (Exception ex)                  {                      Console.WriteLine(ex.Message);                  }              }              else              {                  Console.WriteLine("{0} ({1})", (int)response.StatusCode, response.ReasonPhrase);              }              return products;          }          private static void PrintUnitsInStock(IEnumerable<Product> products)          {              foreach (Product item in products)              {                  Console.Write("Name={0} ; UnitsInStock={1}\r\n", item.ProductName, item.UnitsInStock);              }          }          private static void PrintProducts(IEnumerable<Product> products)          {              foreach (Product item in products)              {                  string toPrint = string.Format(                      "ProductID={0} ProductName={1} SupplierID={2} CategoryID={3} QuantityPerUnit={4} UnitPrice={5} UnitsInStock={6} UnitsOnOrder={7} Discontinued={8} ReorderLevel={9}\r\n\r\n",                       item.ProductID, item.ProductName, item.SupplierID, item.CategoryID, item.QuantityPerUnit, item.UnitPrice, item.UnitsInStock, item.UnitsOnOrder, item.Discontinued, item.ReorderLevel);                  Console.Write(toPrint);              }          } |

Utilise la classe « **HttpClient** » pour envoyer des requêtes HTTP et recevoir des réponses HTTP d'une ressource identifiée par un URI.

##### Client WebForm

|  |
| --- |
| public partial class Form1 : Form  {      public Form1()      {          InitializeComponent();      }      private void Form1\_Load(object sender, EventArgs e)      {          GetData(              ConfigurationManager.AppSettings["adressWebApiServer"].ToString());      }      public class Product      {          public int ProductID { get; set; }          public string ProductName { get; set; }          public int SupplierID { get; set; }          public int CategoryID { get; set; }          public string QuantityPerUnit { get; set; }          public decimal UnitPrice { get; set; }          public Int16 UnitsInStock { get; set; }          public Int16 UnitsOnOrder { get; set; }          public bool Discontinued { get; set; }          public Int16 ReorderLevel { get; set; }      }      private void GetData(string serverAdress)      {          labelServer.Text = serverAdress;          HttpClient client = new HttpClient();          try          {              client.BaseAddress = new Uri(serverAdress);          }          catch (Exception ex)          {              MessageBox.Show(ex.Message);              return;          }          client.DefaultRequestHeaders.Accept.Add(              new MediaTypeWithQualityHeaderValue("application/json"));          HttpResponseMessage response;          try          {              response = client.GetAsync("api/Product").Result;          }          catch (Exception ex )          {              MessageBox.Show(ex.Message);              return;          }          if (response.IsSuccessStatusCode)          {              var products = response.Content.ReadAsAsync<IEnumerable<Product>>().Result;              dataGridView1.DataSource = products;          }          else          {              MessageBox.Show("Error Code" + response.StatusCode + " : Message - " + response.ReasonPhrase);          }      }      private void buttonFetch\_Click(object sender, EventArgs e)      {          dataGridView1.DataSource = null;          GetData(textBoxServerAdress.Text);      }  } |

Utilise la classe « **HttpClient** » pour envoyer des requêtes HTTP et recevoir des réponses HTTP d'une ressource identifiée par un URI.

<http://msdn.microsoft.com/fr-fr/library/ms972974.aspx>

### Sécurité

Le site est sécurisé par l’implémentation des rôles dans ASP.NET Identity 10.6.6.

Les propriétés et méthodes faisant l’objet d’une restriction sont décorées de l’« Annotation »

**[Authorize]**.

Utilisation de l’attribut [ValidateAntiForgeryToken]

Cf. : <http://www.bipinjoshi.net/articles/20e546b4-3ae9-416b-878e-5b12434fe7a6.aspx>

# Déploiement

## Espace de nom.

## Archivage.

## Profils de publication et Web.Config associés

|  |  |
| --- | --- |
| Profils de publication | Fichiers Web.Config associés |
|  | web.config local  web.Development.config (VM locale)  web.Integration.config (VM cdf54)  web.release.config production (HelpInfo)  web.debug.config staging (Ikoula)  web.Test.config (NAS) |

* <http://www.shubho.net/2011/03/transformation-webconfig-aspnet-40.html>
* <http://blogs.msdn.com/b/webdev/archive/2010/10/26/asp-net-web-projects-web-debug-config-amp-web-release-config.aspx>
* <http://blogs.msdn.com/b/webdev/archive/2009/05/04/web-deployment-web-config-transformation.aspx>

## Intégration de la base Identity à la base NorthWind

Donner le même nom à la base de données de la « DefaultConnection » que celle de l’application (NorthWind.mdf ».

1. Activer les Migrations : **PM> Enable-Migrations**

Cela a pour effet de créer une classe Configuration

1. Ajouter une migration : **PM> Add-Migration InitialDatabase**

Cela a pour effet de créer une classe InitialDataBase.

1. Mettre à jour la base de données : **PM> Update-Database**

Cela a pour effet de créer les tables ASP.NET Identity dans la base de l’application Northwind.

## « Migrations » Staging/Production

Réf :

* <http://www.hanselman.com/blog/EntityFrameworkCodeFirstMigrationsAlphaNuGetPackageOfTheWeek10.aspx>
* <http://rdonfack.developpez.com/tutoriels/dotnet/entity-framework-decouverte-code-first-migrations/>

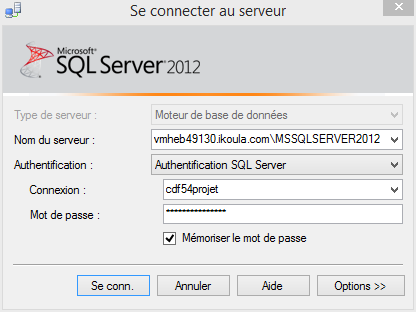
### Script de génération de la base de données

|  |
| --- |
| PM> **Update-Database -Script -SourceMigration:0**  Application de migrations explicites : [201402101412074\_Initialisation].  Application d'une migration explicite : 201402101412074\_Initialisation. |

|  |
| --- |
| DECLARE @CurrentMigration [nvarchar](max)  IF object\_id('[dbo].[\_\_MigrationHistory]') IS NOT NULL  SELECT @CurrentMigration =  (SELECT TOP (1)  [Project1].[MigrationId] AS [MigrationId]  FROM ( SELECT  [Extent1].[MigrationId] AS [MigrationId]  FROM [dbo].[\_\_MigrationHistory] AS [Extent1]  WHERE [Extent1].[ContextKey] = N'Mvc5Ef6WebApiDataFirstNthW.Migrations.Configuration'  ) AS [Project1]  ORDER BY [Project1].[MigrationId] DESC)  IF @CurrentMigration IS NULL  SET @CurrentMigration = '0'  IF @CurrentMigration < '201402101226082\_Initialisation'  BEGIN  CREATE TABLE [dbo].[AspNetRoles] (  [Id] [nvarchar](128) NOT NULL,  [Name] [nvarchar](max) NOT NULL,  CONSTRAINT [PK\_dbo.AspNetRoles] PRIMARY KEY ([Id])  )  CREATE TABLE [dbo].[AspNetUsers] (  [Id] [nvarchar](128) NOT NULL,  [UserName] [nvarchar](max),  [PasswordHash] [nvarchar](max),  [SecurityStamp] [nvarchar](max),  [FirstName] [nvarchar](max),  [LastName] [nvarchar](max),  [Email] [nvarchar](max),  [Discriminator] [nvarchar](128) NOT NULL,  CONSTRAINT [PK\_dbo.AspNetUsers] PRIMARY KEY ([Id])  )  CREATE TABLE [dbo].[AspNetUserClaims] (  [Id] [int] NOT NULL IDENTITY,  [ClaimType] [nvarchar](max),  [ClaimValue] [nvarchar](max),  [User\_Id] [nvarchar](128) NOT NULL,  CONSTRAINT [PK\_dbo.AspNetUserClaims] PRIMARY KEY ([Id])  )  CREATE INDEX [IX\_User\_Id] ON [dbo].[AspNetUserClaims]([User\_Id])  CREATE TABLE [dbo].[AspNetUserLogins] (  [UserId] [nvarchar](128) NOT NULL,  [LoginProvider] [nvarchar](128) NOT NULL,  [ProviderKey] [nvarchar](128) NOT NULL,  CONSTRAINT [PK\_dbo.AspNetUserLogins] PRIMARY KEY ([UserId], [LoginProvider], [ProviderKey])  )  CREATE INDEX [IX\_UserId] ON [dbo].[AspNetUserLogins]([UserId])  CREATE TABLE [dbo].[AspNetUserRoles] (  [UserId] [nvarchar](128) NOT NULL,  [RoleId] [nvarchar](128) NOT NULL,  CONSTRAINT [PK\_dbo.AspNetUserRoles] PRIMARY KEY ([UserId], [RoleId])  )  CREATE INDEX [IX\_RoleId] ON [dbo].[AspNetUserRoles]([RoleId])  CREATE INDEX [IX\_UserId] ON [dbo].[AspNetUserRoles]([UserId])  ALTER TABLE [dbo].[AspNetUserClaims] ADD CONSTRAINT [FK\_dbo.AspNetUserClaims\_dbo.AspNetUsers\_User\_Id] FOREIGN KEY ([User\_Id]) REFERENCES [dbo].[AspNetUsers] ([Id]) ON DELETE CASCADE  ALTER TABLE [dbo].[AspNetUserLogins] ADD CONSTRAINT [FK\_dbo.AspNetUserLogins\_dbo.AspNetUsers\_UserId] FOREIGN KEY ([UserId]) REFERENCES [dbo].[AspNetUsers] ([Id]) ON DELETE CASCADE  ALTER TABLE [dbo].[AspNetUserRoles] ADD CONSTRAINT [FK\_dbo.AspNetUserRoles\_dbo.AspNetRoles\_RoleId] FOREIGN KEY ([RoleId]) REFERENCES [dbo].[AspNetRoles] ([Id]) ON DELETE CASCADE  ALTER TABLE [dbo].[AspNetUserRoles] ADD CONSTRAINT [FK\_dbo.AspNetUserRoles\_dbo.AspNetUsers\_UserId] FOREIGN KEY ([UserId]) REFERENCES [dbo].[AspNetUsers] ([Id]) ON DELETE CASCADE  CREATE TABLE [dbo].[\_\_MigrationHistory] (  [MigrationId] [nvarchar](150) NOT NULL,  [ContextKey] [nvarchar](300) NOT NULL,  [Model] [varbinary](max) NOT NULL,  [ProductVersion] [nvarchar](32) NOT NULL,  CONSTRAINT [PK\_dbo.\_\_MigrationHistory] PRIMARY KEY ([MigrationId], [ContextKey])  )  INSERT [dbo].[\_\_MigrationHistory]([MigrationId], [ContextKey], [Model], [ProductVersion])  VALUES (N'201402101226082\_Initialisation', N'Mvc5Ef6WebApiDataFirstNthW.Migrations.Configuration', |

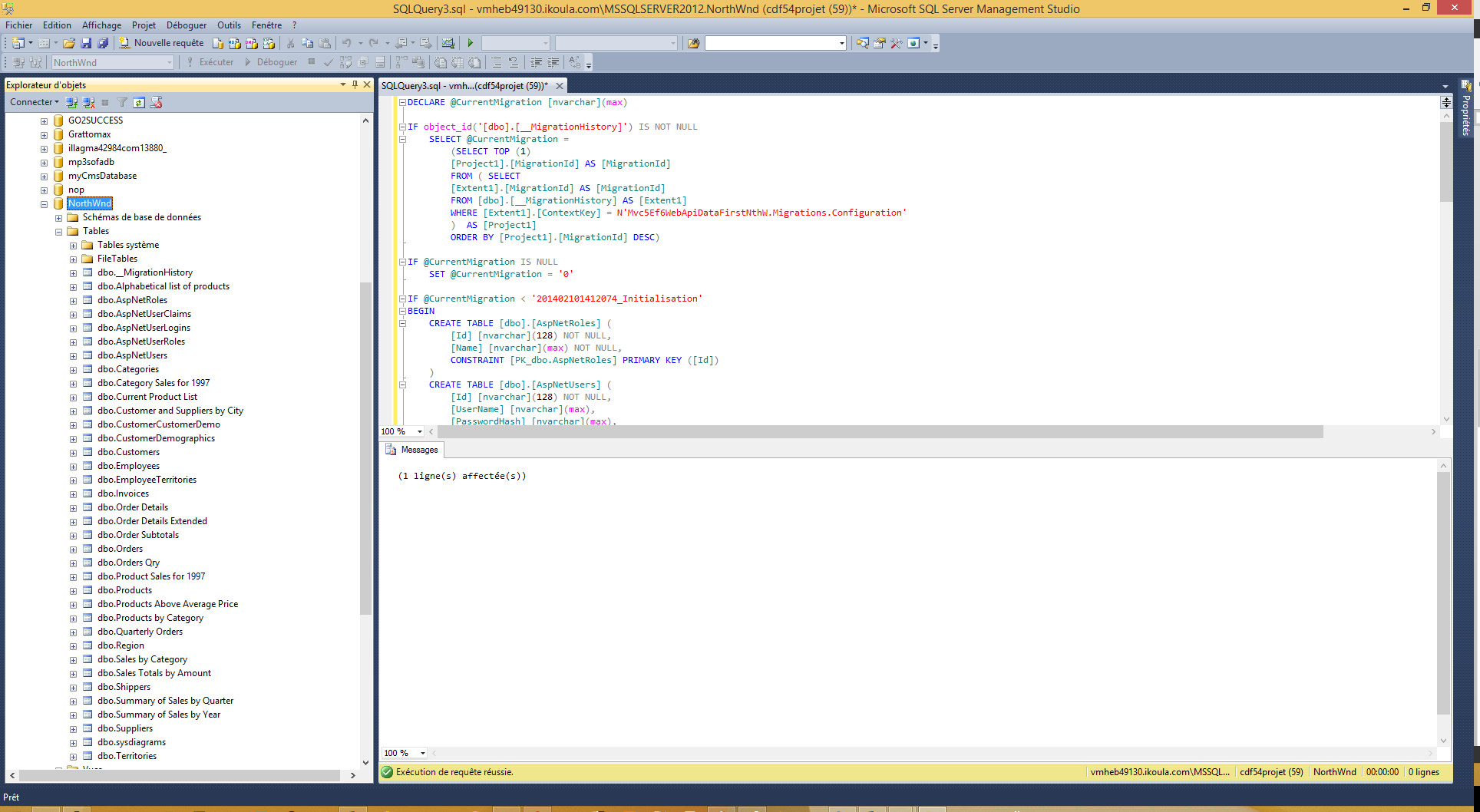
Enregistré sous le répertoire « **Migrations** », en « **Initialisation Data Base.sql** »

### Migration « Initialisation » sur le serveur « Staging » [IKOULA]



Ouverture d’un script SQL, et copier/coller le contenu de « **Initialisation Data Base.sql** »

Dans la fenêtre de script de SQL Server.



Cliquer sur « **exécuter** ».

Même opération sue le serveur de Production

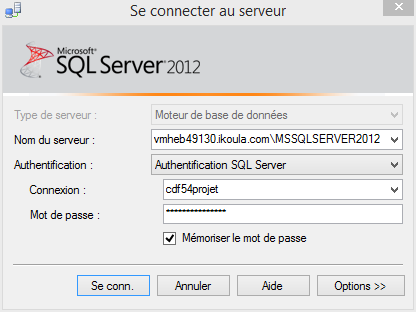
### Script de génération de mise à jour pour l’ajout de photo

|  |
| --- |
| PM> **Update-Database -Script -SourceMigration:201402101412074\_Initialisation** TargetMigration:201402111528351\_Photo  Application de migrations explicites : [201402111528351\_Photo].  Application d'une migration explicite : 201402111528351\_Photo. |

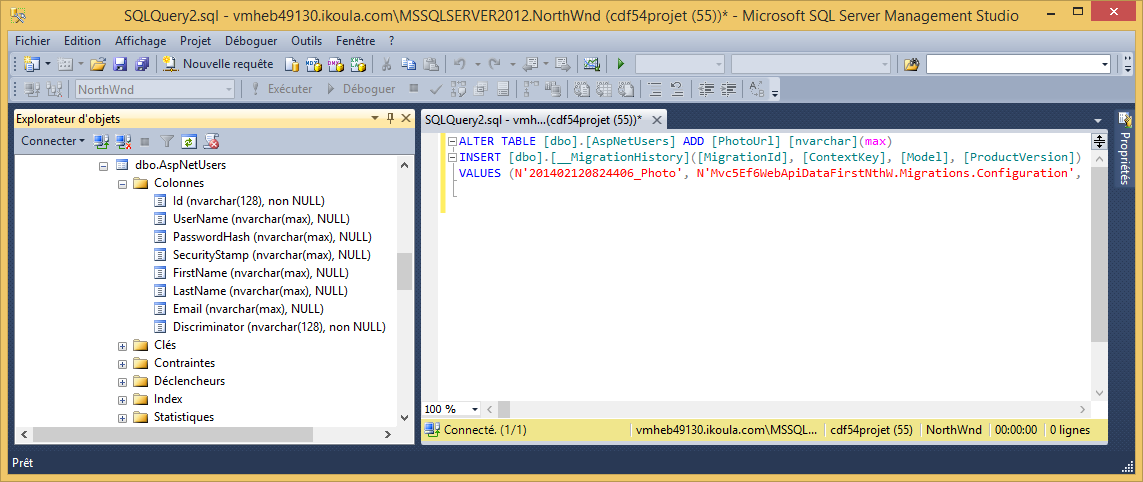
|  |
| --- |
| ALTER TABLE [dbo].[AspNetUsers] ADD [PhotoUrl] [nvarchar](max)  INSERT [dbo].[\_\_MigrationHistory]([MigrationId], [ContextKey], [Model], [ProductVersion])  VALUES (N'201402120824406\_Photo', N'Mvc5Ef6WebApiDataFirstNthW.Migrations.Configuration',   , N'6.0.2-21211') |

Enregistré sous le répertoire « **Migrations** », en « **Ajout Photo.sql** »

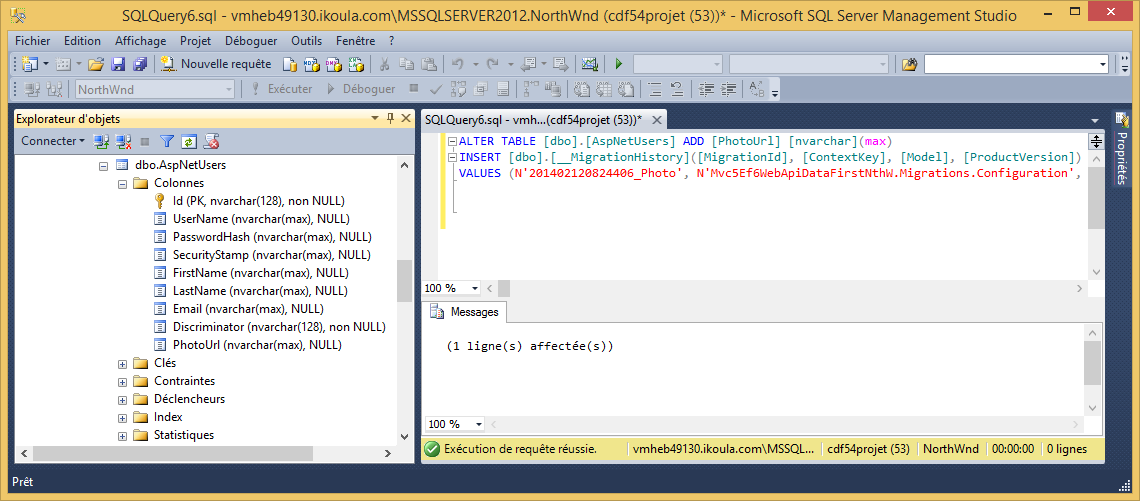
### Migration « Photo » sur le serveur « Staging » [IKOULA]



Avant exécution



Après exécution



Même opération sue le serveur de Production

# Recommandations

[Liste des recommandations à suivre tout au long du projet.]

* 1. **Instructions relatives aux noms.**

*Langue Anglaise exclusivement !*

Pour les variables, propriétés, méthodes, commentaires… utilisez exclusivement la langue Anglaise.

*Casse*

* Utiliser dans tous les cas la casse Pascal  
  Ex : GetUsers
* Pour les paramètres utiliser la casse Camel  
  Ex : userName

Utiliser le mot clef "this" lorsque vous faite référence à un champ, une propriété ou une méthode dans la classe.

* Réf :
  + *Submain\_DotNET\_Coding\_Guidelines.pdf*
  + *cs-coding-standard-bellware.pdf*

*Ressources et Settings*

Ne pas coder en dur les chaînes de caractères, des valeurs numériques. Utiliser le fichier de ressources ou le fichier de paramètres (settings).

* 1. **Conventions de dénomination du projet.**
     1. **Sous-systèmes.**

|  |  |  |
| --- | --- | --- |
| **Catégorie** | **Nom** | **SIGLE** |
| Nom du projet | ***NorthWind54*** |  |
| Sous-système 1 |  |  |
| Sous-système 2 |  |  |
| Sous-système 3 |  |  |
| Sous-système 4 |  |  |

## Ressources et Settings

Ne pas coder en dur les chaînes de caractères, des valeurs numériques. Utiliser le fichier de ressources ou le fichier de paramètres (settings).

## Rédaction des dossiers

Adapter la couleur de l’entête des tableaux et décorer les paragraphes en fonction du numéro Étudiant qui vous a été attribué

# Journal de bord

|  | **JOURNAL DE BORD : ETUDIANT x [sigle]** | | |
| --- | --- | --- | --- |
| **Date** | | **Intervenants** | **Objet** |
|  | |  |  |
|  | |  |  |

|  | **JOURNAL DE BORD : ETUDIANT x [sigle]** | | |
| --- | --- | --- | --- |
| **Date** | | **Intervenants** | **Objet** |
|  | |  |  |
|  | |  |  |

|  | **JOURNAL DE BORD : ETUDIANT x [sigle]** | | |
| --- | --- | --- | --- |
| **Date** | | **Intervenants** | **Objet** |
|  | |  |  |
|  | |  |  |

|  | **JOURNAL DE BORD : ETUDIANT x [sigle]** | | |
| --- | --- | --- | --- |
| **Date** | | **Intervenants** | **Objet** |
|  | |  |  |
|  | |  |  |

NOTES