

## Table des matières

Présentation .....	5
Modèle de conception MVC.....	5
Prise en charge du développement axé sur des tests.....	7
Quand créer une application MVC .....	7
Avantages d'une application Web basée sur MVC.....	7
Avantages d'une application Web basée sur des Web Forms .....	8
Fonctionnalités de l'infrastructure ASP.NET MVC.....	9
Gestion des packages à l'aide de la Console Package Manager.....	10
Lister les packages installables dans un projet.....	11
Rechercher un package .....	12
Installer un package.....	12
Lister les packages installés dans un projet .....	13
Désinstaller un package.....	14
Mettre à jour un package .....	14
Création de projet ASP.NET MVC .....	15
Création d'une application ASP.NET MVC sous Visual Studio.....	16
1. Création d'un site .....	16
2. Organisation des répertoires.....	19
3. Création du modèle.....	19
4. Définition du contrôleur.....	22
5. Ajout des vues .....	23
Définition des routes .....	25
1. D'une action à l'autre .....	26
2. Mise à jour du modèle et redirection.....	32
3. Validation.....	33
Le moteur de rendu Razor et les vues.....	34
1. La syntaxe C# dans les vues CSHTML .....	34
2. Structure et organisation des vues.....	39
Définir des zones (areas) .....	42
Mise en route du CodeFirst.....	43

---

---

## MVC ASP.NET

---

Chaine de connexion .....	43
Context .....	44
CodeFirst.....	45
Mise à Jour de la base .....	46
Le Model.....	48
Notion sur les DataAnnotations .....	48
System.ComponentModel.DataAnnotations Attributes:.....	48
System.ComponentModel.DataAnnotations.Schema Attributes:.....	48
Utilisation de validation Annotations.....	49
Required .....	49
StringLength .....	49
RegularExpression .....	49
Range.....	49
Compare .....	50
Remote .....	50
DataType Enumeration.....	50
Gestion Etablissement élémentaire .....	52
Module Inscription .....	52
Module Facturation.....	60
Module Gestion des notes .....	60
Les contrôleurs .....	60
Ajout d'un nouveau contrôleur .....	61
Méthode d'action.....	63
Méthode d'action par défaut .....	64
ActionResult .....	64
Méthode d'action Paramètres .....	66
Sélecteurs action .....	67
ActionName.....	67
NonAction.....	67
ActionVerbs .....	68
View .....	71
Razor view .....	72
Créer une nouvelle View .....	73

---

---

## MVC ASP.NET

---

Razor Syntax .....	78
Inline expression.....	78
Multi-statement Code block.....	78
Display text from code block.....	79
if-else condition.....	80
for loop .....	80
Model .....	80
Declare Variables.....	81
HTML Helpers .....	81
Create TextBox using HtmlHelper .....	83
Create TextArea using HtmlHelper.....	85
Create CheckBox using HtmlHelper .....	86
Create RadioButton using HtmlHelper.....	87
Create DropDownList using HtmlHelper .....	88
Create Hidden field using HtmlHelper.....	90
Create Html String using HtmlHelper.....	91
Create Label using HtmlHelper.....	92
HtmlHelper.Editor .....	93
Pratique controller/ View avec gestion etablissement Elementaire. ....	94
Installation et mise en œuvre des pagedList.....	100
Recherche sur une page .....	102
AJAX .....	103
Employee.cs Code .....	106
Controller vide.....	106
View Employee .....	107
Javascript Employee .....	108
Dynamic Load Data.....	111
Fichier de ressource .....	111
Notification provenant du Controller.....	115
Gestion des erreurs .....	120
Impression .....	120
La création de la classe.....	120
La création du rapport.....	120

---

---

## MVC ASP.NET

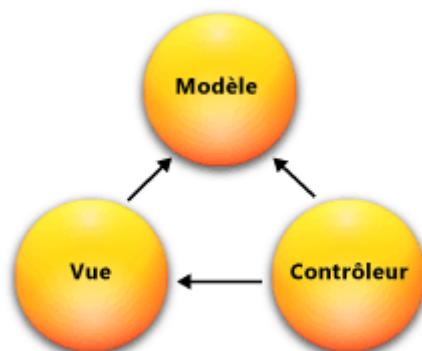
---

La méthode d'appel.....	122
Securite.....	124
Authentification.....	127
Authentication and Authorization	Authentication .....
Building the Sample.....	129
Create a Database.....	130

### Présentation

Le modèle architectural MVC (Model-View-Controller) sépare une application en trois composants principaux : le modèle, la vue et le contrôleur. L'infrastructure ASP.NET MVC offre une alternative au modèle Web Forms ASP.NET pour la création d'applications Web. ASP.NET MVC est une infrastructure de présentation simple et facilement testable, qui (comme celle des applications utilisant des Web Forms) est intégrée aux fonctionnalités ASP.NET existantes, telles que les pages maîtres et l'authentification basée sur l'appartenance. L'infrastructure MVC est définie dans l'assembly System.Web.Mvc.

### Modèle de conception MVC



MVC est un modèle de conception standard qui est connu par de nombreux développeurs. Certains types d'applications Web tireront parti de l'infrastructure MVC. D'autres continueront à utiliser le modèle d'application ASP.NET traditionnel qui est basé sur les Web Forms et les publications (postbacks). D'autres encore combineront les deux approches, l'une n'excluant pas l'autre.

L'infrastructure MVC inclut les composants suivants :

- **Modèles.** Les objets de modèle sont les parties de l'application qui implémentent la logique du domaine de données de l'application. Souvent, ils récupèrent l'état du modèle et le stockent dans une base de

données. Par exemple, un objet Product peut récupérer des informations dans une base de données, les exploiter, puis réécrire les informations mises à jour dans une table Products d'une base de données SQL Server. Dans les petites applications, le modèle est souvent une séparation conceptuelle plutôt qu'une séparation physique. Par exemple, si l'application sert uniquement à lire un groupe de données et à l'envoyer à la vue, elle ne comporte pas de couche de modèle physique, ni de classe associée. Dans ce cas, le groupe de données joue le rôle d'un objet de modèle.

- **Vues.** Les vues sont les composants qui affichent l'interface utilisateur (IU) de l'application. En général, cette interface utilisateur est créée à partir des données du modèle. Il peut s'agir par exemple d'une vue d'édition d'une table Products affichant des zones de texte, des listes déroulantes et des cases à cocher en fonction de l'état actuel d'un objet Product.
- **Contrôleurs.** Les contrôleurs sont les composants qui gèrent les interventions de l'utilisateur, exploitent le modèle et finalement sélectionnent une vue permettant de restituer l'interface utilisateur. Dans une application MVC, la vue sert uniquement à afficher les informations ; le contrôleur gère les entrées et interactions de l'utilisateur, et y répond. Par exemple, il gère les valeurs de chaîne de requête et les passe au modèle, qui peut à son tour les utiliser pour interroger la base de données.

Le modèle MVC vous aide à créer des applications qui séparent les différents aspects de l'application (logique d'entrée, logique métier et logique de l'interface utilisateur) en assurant un couplage lâche entre ces éléments. Le modèle spécifie l'emplacement où chaque genre de logique doit figurer dans l'application. La logique de l'interface utilisateur appartient à la vue. La logique d'entrée appartient au contrôleur. La logique métier appartient au modèle. Cette séparation vous aide à gérer la complexité lorsque vous gérez une application car elle vous permet de vous concentrer sur un aspect de l'implémentation à la fois. Par exemple, vous pouvez vous concentrer sur la vue sans dépendre de la logique métier.

---

## MVC ASP.NET

---

Le couplage lâche entre les trois principaux composants d'une application MVC favorise également le développement en parallèle. Par exemple, un premier développeur peut se concentrer sur la vue, un deuxième sur la logique du contrôleur et un troisième sur la logique métier du modèle.

### Prise en charge du développement axé sur des tests

Le modèle MVC permet non seulement de gérer la complexité, mais également de simplifier le processus de test comparativement à celui d'une application Web ASP.NET basée sur des Web Forms. Par exemple, dans une application Web ASP.NET basée sur des Web Forms, une même classe est utilisée pour afficher la sortie et répondre aux interventions de l'utilisateur. L'écriture de tests automatisés pour les applications ASP.NET basées sur des Web Forms peut s'avérer complexe, car pour tester une page individuelle, vous devez instancier la classe de la page, tous ses contrôles enfants et les autres classes dépendantes dans l'application. Étant donné que de nombreuses classes sont instanciées pour exécuter la page, il peut s'avérer difficile d'écrire des tests centrés exclusivement sur des parties individuelles de l'application. Par conséquent, les tests des applications ASP.NET basées sur des Web Forms peuvent être plus complexes à implémenter que les tests d'une application MVC. En outre, les tests d'une application ASP.NET basée sur des Web Forms requièrent un serveur Web. L'infrastructure MVC dissocie les composants et utilise de façon intensive les interfaces, ce qui permet de tester des composants individuels en les isolant du reste de l'infrastructure.

### Quand créer une application MVC

Vous devez déterminer avec soin s'il convient d'implémenter une application Web à l'aide de l'infrastructure ASP.NET MVC ou du modèle Web Forms ASP.NET. L'infrastructure MVC ne remplace pas le modèle Web Forms ; vous pouvez utiliser l'une ou l'autre des infrastructures pour les applications Web. (Si certaines de vos applications sont basées sur des Web Forms, elles continuent à fonctionner exactement comme elles l'ont toujours fait.)

Avant de décider d'utiliser l'infrastructure MVC ou le modèle Web Forms pour un site Web spécifique, évaluez les avantages de chaque approche.

### Avantages d'une application Web basée sur MVC

L'infrastructure ASP.NET MVC offre les avantages suivants :

- Elle permet de gérer plus facilement la complexité en décomposant une application en modèle, vue et contrôleur.
- Elle n'utilise pas l'état d'affichage ni les formulaires serveur. L'infrastructure MVC s'avère par conséquent idéale pour les développeurs souhaitant contrôler totalement le comportement d'une application.
- Elle utilise un modèle de contrôleur frontal qui traite les requêtes de l'application Web par le biais d'un contrôleur unique. De cette façon, vous pouvez concevoir une application prenant en charge une infrastructure de routage complète. Pour plus d'informations, consultez Front Controller (Contrôleur frontal).
- Elle offre une meilleure prise en charge du développement axé sur des tests.
- Elle est parfaitement adaptée aux applications Web qui sont prises en charge par de grandes équipes de développeurs et aux concepteurs Web qui ont besoin d'un haut niveau de contrôle sur le comportement des applications.

### Avantages d'une application Web basée sur des Web Forms

L'infrastructure basée sur des Web Forms offre les avantages suivants :

- Elle prend en charge un modèle d'événement permettant la conservation de l'état via HTTP, mécanisme qui s'avère particulièrement utile pour le développement d'applications Web métier. L'application basée sur des Web Forms fournit des dizaines d'événements pris en charge dans des centaines de contrôles serveur.
- Elle utilise un modèle de contrôleur de pages qui ajoute des fonctionnalités aux pages individuelles. Pour plus d'informations, consultez Page Controller (Contrôleur de pages).
- Elle utilise l'état d'affichage et les formulaires serveur, ce qui peut faciliter la gestion des informations d'état.
- Elle est parfaitement adaptée aux petites équipes de concepteurs et de développeurs Web qui souhaitent tirer parti des nombreux composants disponibles pour le développement rapide d'applications.
- En général, elle s'avère moins complexe pour le développement d'applications car les composants (la classe Page, les contrôles, etc.) sont

étroitement intégrés et requièrent habituellement moins de code que le modèle MVC.

### Fonctionnalités de l'infrastructure ASP.NET MVC

L'infrastructure ASP.NET MVC intègre les fonctionnalités suivantes :

- Séparation des tâches d'application (logique d'entrée, logique métier et logique de l'interface utilisateur), testabilité et développement axé sur des tests. Tous les contrats de base de l'infrastructure MVC sont basés sur des interfaces et peuvent être testés à l'aide d'objets fictifs qui simulent le comportement d'objets réels de l'application. Les tests unitaires de l'application sont à la fois rapides et flexibles étant donné que vous pouvez les effectuer sans avoir à exécuter les contrôleurs dans un processus ASP.NET. Vous pouvez utiliser toute infrastructure de test unitaire compatible avec .NET Framework.
- Infrastructure extensible et en fichable. Les composants de l'infrastructure ASP.NET MVC sont conçus de façon à pouvoir être facilement remplacés ou personnalisés. Vous pouvez incorporer vos propres moteurs d'affichage, stratégie de routage des URL et sérialisation des paramètres de méthode d'action, ainsi que d'autres composants. L'infrastructure ASP.NET MVC prend en charge également l'utilisation des modèles de conteneur Injection de dépendances et Inversion de contrôle. L'injection de dépendances vous permet d'injecter des objets dans une classe, au lieu de faire appel à la classe pour qu'elle crée les objets eux-mêmes. L'inversion de contrôle spécifie que si un objet requiert un autre objet, le premier objet doit obtenir le deuxième à partir d'une source extérieure, telle qu'un fichier de configuration. Ce processus facilite les tests.
- Prise en charge complète du routage ASP.NET, qui est un composant de mappage d'URL puissant vous permettant de générer des applications dont les URL sont compréhensibles et peuvent faire l'objet de recherche. Les URL ne doivent pas nécessairement inclure des extensions de nom de fichier et sont conçues pour prendre en charge des modèles de dénomination d'URL adaptés à l'optimisation des moteurs de recherche (SEO, Search Engine Optimization) et à l'adressage au format REST (Representational State Transfer).

---

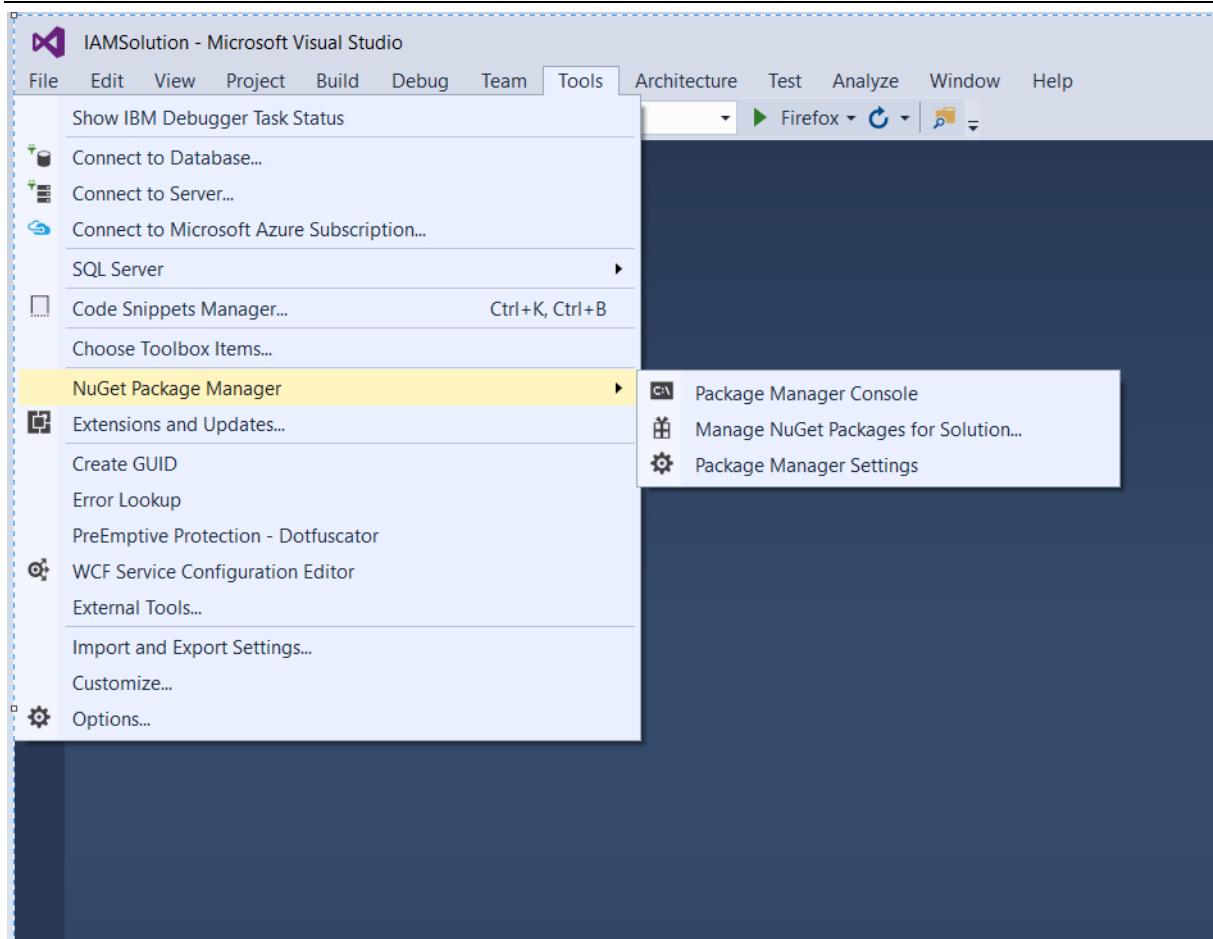
## MVC ASP.NET

---

- Prise en charge du balisage des fichiers de balisage de page ASP.NET (fichiers .aspx), de contrôle utilisateur (fichiers .ascx) et de page maître (fichiers .master) existants en tant que modèles de vue. Vous pouvez utiliser les fonctionnalités ASP.NET existantes avec l'infrastructure ASP.NET MVC, à savoir les pages maîtres imbriquées, les expressions en ligne (`<%= %>`), les contrôles serveur déclaratifs, les modèles, les liaisons de données, la localisation, etc.
- Prise en charge des fonctionnalités ASP.NET existantes. ASP.NET MVC vous permet d'utiliser des fonctionnalités telles que l'authentification par formulaire et l'authentification Windows, l'autorisation d'URL, l'appartenance et les rôles, la mise en cache de sortie et de données, la gestion d'état de session et de profil, le contrôle d'état, le système de configuration et l'architecture du fournisseur.

### Gestion des packages à l'aide de la Console Package Manager

Cette rubrique décrit comment trouver, installer, supprimer et mettre à jour NuGet paquets en utilisant les commandes PowerShell. Vous pouvez également travailler avec des paquets en utilisant la boîte de dialogue NuGet Gérer les forfaits. Pour plus d'informations.



## Lister les packages installables dans un projet

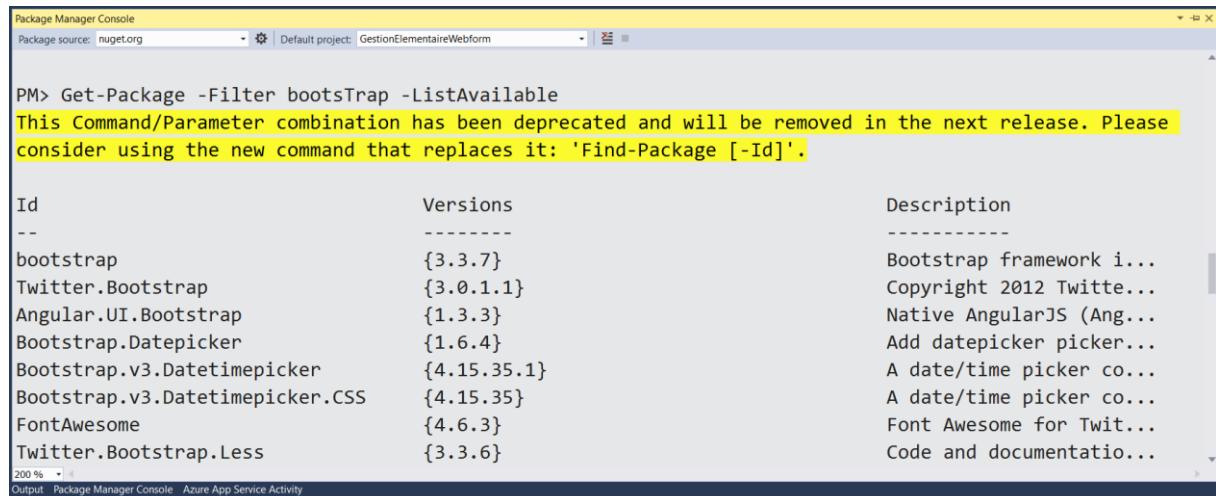
```
Package Manager Console
Package source: nuget.org | Default project: GestionElementaireWebform | X
Package Manager Console Host Version 3.4.0.798

Type 'get-help NuGet' to see all available NuGet commands.

PM> Get-Package -ListAvailable

Id          Versions      Description
--          -----
Newtonsoft.Json {9.0.1}    Json.NET is a popular...
EntityFramework {6.1.3}   Entity Framework is M...
NUnit        {3.4.1}    NUnit is a unit-testi...
jQuery       {3.1.0}    jQuery is a new kind ...
bootstrap    {3.3.7}    Bootstrap framework i...
AutoMapper   {5.1.1}    A convention-based ob...
Microsoft.AspNet.Mvc {5.2.3}  This package contains...
```

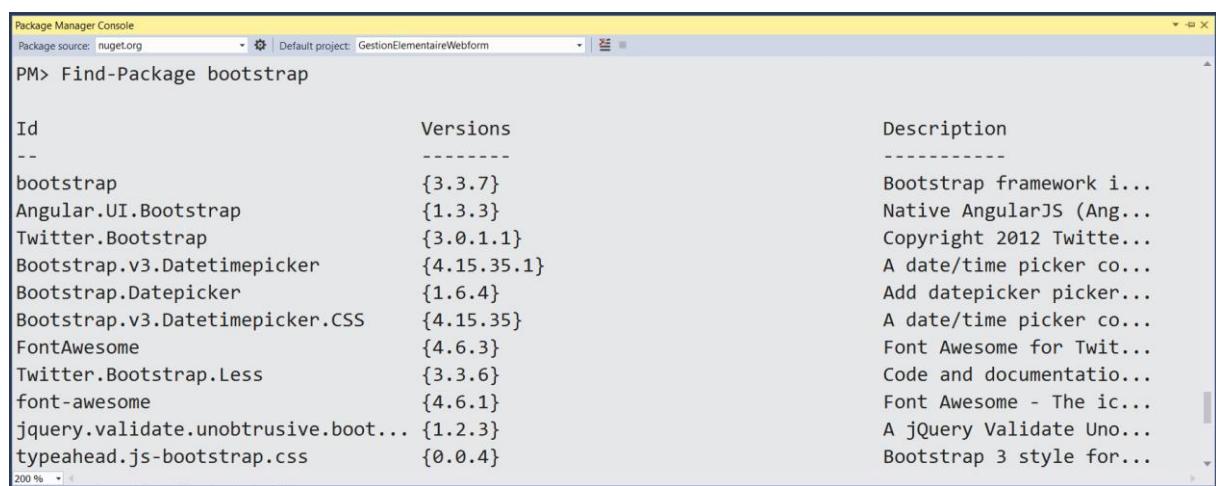
## Rechercher un package



```
PM> Get-Package -Filter bootsTrap -ListAvailable
This Command/Parameter combination has been deprecated and will be removed in the next release. Please consider using the new command that replaces it: 'Find-Package [-Id]'.

Id          Versions      Description
--          -----
bootstrap    {3.3.7}       Bootstrap framework i...
Twitter.Bootstrap {3.0.1.1}  Copyright 2012 Twitte...
Angular.UI.Bootstrap {1.3.3}  Native AngularJS (Ang...
Bootstrap.Datepicker {1.6.4}   Add datepicker picker...
Bootstrap.v3.Datetimepicker {4.15.35.1} A date/time picker co...
Bootstrap.v3.Datetimepicker.CSS {4.15.35}  A date/time picker co...
FontAwesome    {4.6.3}       Font Awesome for Twit...
Twitter.Bootstrap.Less {3.3.6}  Code and documentatio...
```

Output: Package Manager Console Azure App Service Activity

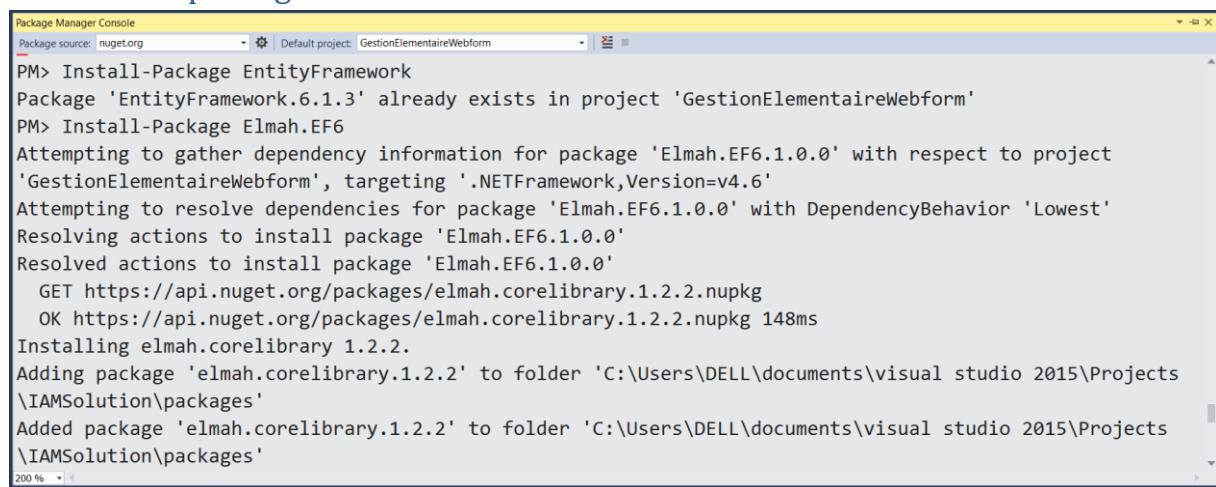


```
PM> Find-Package bootstrap

Id          Versions      Description
--          -----
bootstrap    {3.3.7}       Bootstrap framework i...
Angular.UI.Bootstrap {1.3.3}  Native AngularJS (Ang...
Twitter.Bootstrap {3.0.1.1}  Copyright 2012 Twitte...
Bootstrap.v3.Datetimepicker {4.15.35.1} A date/time picker co...
Bootstrap.Datepicker {1.6.4}   Add datepicker picker...
Bootstrap.v3.Datetimepicker.CSS {4.15.35}  A date/time picker co...
FontAwesome    {4.6.3}       Font Awesome for Twit...
Twitter.Bootstrap.Less {3.3.6}  Code and documentatio...
font-awesome {4.6.1}       Font Awesome - The ic...
jquery.validate.unobtrusive.boot... {1.2.3}  A jQuery Validate Uno...
typeahead.js-bootstrap.css {0.0.4}  Bootstrap 3 style for...
```

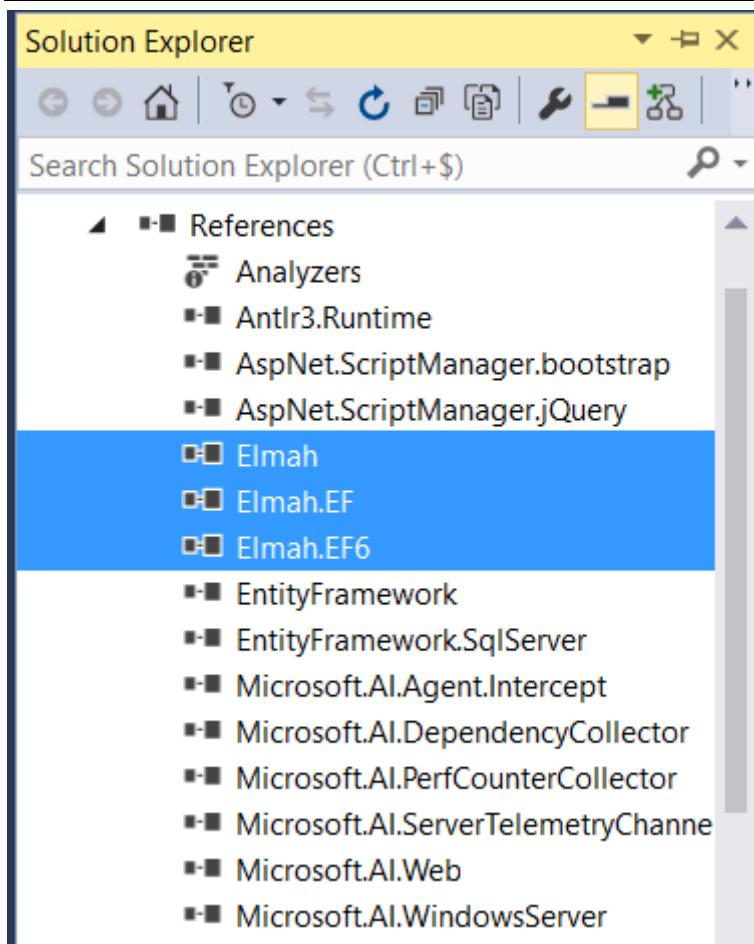
200 %

## Installer un package



```
PM> Install-Package EntityFramework
Package 'EntityFramework.6.1.3' already exists in project 'GestionElementaireWebform'
PM> Install-Package Elmah.EF6
Attempting to gather dependency information for package 'Elmah.EF6.1.0.0' with respect to project
'GestionElementaireWebform', targeting '.NETFramework,Version=v4.6'
Attempting to resolve dependencies for package 'Elmah.EF6.1.0.0' with DependencyBehavior 'Lowest'
Resolving actions to install package 'Elmah.EF6.1.0.0'
Resolved actions to install package 'Elmah.EF6.1.0.0'
  GET https://api.nuget.org/packages/elmah.corelibrary.1.2.2.nupkg
  OK https://api.nuget.org/packages/elmah.corelibrary.1.2.2.nupkg 148ms
Installing elmah.corelibrary 1.2.2.
Adding package 'elmah.corelibrary.1.2.2' to folder 'C:\Users\DELL\documents\visual studio 2015\Projects
\IAMSolution\packages'
Added package 'elmah.corelibrary.1.2.2' to folder 'C:\Users\DELL\documents\visual studio 2015\Projects
\IAMSolution\packages'
```

200 %



### Lister les packages installés dans un projet

```
PM> Get-Package
```

Id	Versions	ProjectName
Antlr	{3.4.1.9004}	GestionElementaireWeb...
AspNet.ScriptManager.bootstrap	{3.0.0}	GestionElementaireWeb...
AspNet.ScriptManager.jQuery	{1.10.2}	GestionElementaireWeb...
bootstrap	{3.0.0}	GestionElementaireWeb...
elmah.corelibrary	{1.2.2}	GestionElementaireWeb...
Elmah.EF	{1.0.0.0}	GestionElementaireWeb...
Elmah.EF6	{1.0.0.0}	GestionElementaireWeb...
EntityFramework	{6.1.3}	GestionElementaireWeb...
jQuery	{1.10.2}	GestionElementaireWeb...
Microsoft.ApplicationInsights	{1.2.3}	GestionElementaireWeb...
Microsoft.ApplicationInsights.Ag...	{1.2.0}	GestionElementaireWeb...

## Désinstaller un package

```
Package Manager Console
Package source: nuget.org | Default project: GestionElementaireWebform
PM> Uninstall-Package Elmah.EF6
Attempting to gather dependency information for package 'Elmah.EF6.1.0.0' with respect to project
'GestionElementaireWebform', targeting '.NETFramework,Version=v4.6'
Resolving actions to uninstall package 'Elmah.EF6.1.0.0'
Resolved actions to uninstall package 'Elmah.EF6.1.0.0'
Removed package 'Elmah.EF6.1.0.0' from 'packages.config'
Successfully uninstalled 'Elmah.EF6.1.0.0' from GestionElementaireWebform
Removing package 'Elmah.EF6.1.0.0' from folder 'C:\Users\DELL\documents\visual studio 2015\Projects
\IAMSolution\packages'
Removed package 'Elmah.EF6.1.0.0' from folder 'C:\Users\DELL\documents\visual studio 2015\Projects
\IAMSolution\packages'
PM> |
```

## Mettre à jour un package

Id	Versions	Description
--	-----	-----
Antlr	{3.5.0.2}	ANother Tool for Lang...
AspNet.ScriptManager.bootstrap	{3.3.6}	This package contains...
AspNet.ScriptManager.jQuery	{3.1.0}	This package contains...
bootstrap	{3.3.7}	Bootstrap framework i...
jQuery	{3.1.0}	jQuery is a new kind ...
Microsoft.ApplicationInsights	{2.1.0}	Application Insights ...
Microsoft.ApplicationInsights.Ag...	{2.0.1}	This NuGet enables In...
Microsoft.ApplicationInsights.De...	{2.1.0}	Application Insights ...
Microsoft.ApplicationInsights.Pe...	{2.1.0}	Application Insights ...
Microsoft.ApplicationInsights.Web	{2.1.0}	Application Insights ...
Microsoft.ApplicationInsights.Wi...	{2.1.0}	Application Insights ...

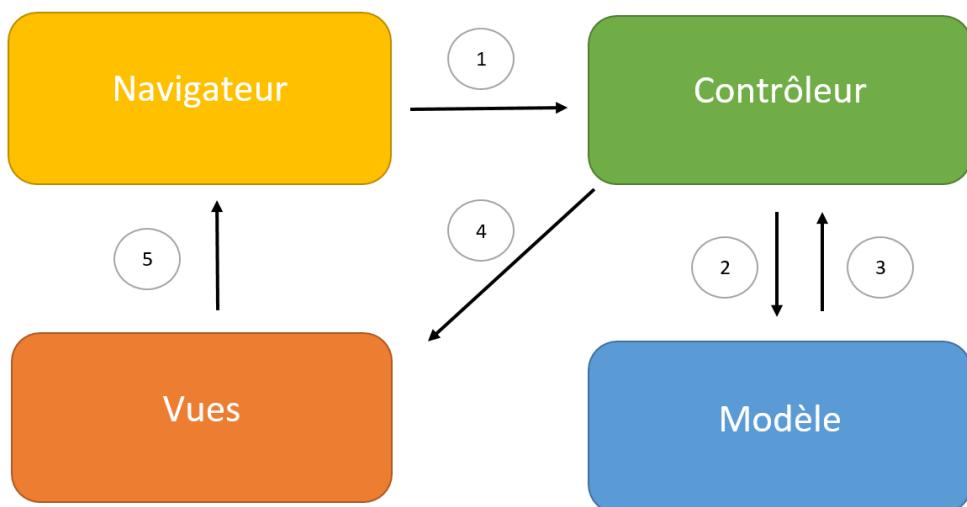
```
Package Manager Console
Package source: nuget.org | Default project: GestionElementaireWebform
PM> Update-Package jquery
Attempting to gather dependency information for multiple packages with respect to project
'GestionElementaireWebform', targeting '.NETFramework,Version=v4.6'
Attempting to resolve dependencies for multiple packages.
Resolving actions install multiple packages
Removed package 'AspNet.ScriptManager.jQuery 1.10.2' from 'packages.config'
Successfully uninstalled 'AspNet.ScriptManager.jQuery 1.10.2' from GestionElementaireWebform
Executing script file 'C:\Users\DELL\documents\visual studio 2015\Projects\IAMSolution\packages
\jQuery.1.10.2\Tools\uninstall.ps1'
Removed package 'jQuery 1.10.2' from 'packages.config'
Successfully uninstalled 'jQuery 1.10.2' from GestionElementaireWebform
  GET https://api.nuget.org/v3-flatcontainer/jquery/3.1.0/jquery.3.1.0.nupkg
  OK https://api.nuget.org/v3-flatcontainer/jquery/3.1.0/jquery.3.1.0.nupkg 627ms
Installing jQuery 3.1.0.
Adding package 'jQuery.3.1.0' to folder 'C:\Users\DELL\documents\visual studio 2015\Projects\IAMSolution
200 % |
```

### Création de projet ASP.NET MVC

Notre éditeur sera le Visual studio. Le langage sera du csharp.

MVC est un patron de conception (ou design pattern en anglais) désormais très répandu pour réaliser des sites web. Il a tout d'abord été conçu pour des applications dites « client lourd », c'est-à-dire dont la majorité des données sont traitées sur le poste client. Mais ce modèle de conception était tellement puissant qu'il a finalement été adopté comme modèle pour la création d'applications ou de sites web. Ce design pattern est une solution reconnue permettant de séparer l'affichage des informations (la vue), les actions de l'utilisateur (le contrôleur) et l'accès aux données (le modèle). Il s'agit du principal avantage apporté par ce modèle de conception, qui présente donc une architecture claire et normalisée, qui facilitera également le développement, l'évolutivité et la maintenance des projets. MVC signifie donc Modèle-Vue-Contrôleur.

Le modèle est ce que l'on appelle un objet « métier », c'est le cœur de l'application. Il traite principalement les données et les interactions avec la base de données. La vue traite ce que nous voyons dans notre navigateur web, elle restitue le modèle au sein de notre interface web et permet à l'utilisateur d'interagir avec le modèle. Le contrôleur fait le lien entre le modèle et la vue, il gère les requêtes des utilisateurs et détermine les traitements qui doivent être effectués pour chaque action. Il va utiliser les données du modèle, les traiter et les envoyer à la vue pour que celle-ci les affiche.



Légende du schéma :

- 1 - L'utilisateur envoie une requête HTTP
- 2 - Le contrôleur appelle le modèle, celui-ci va récupérer les données
- 3 - Le modèle retourne les données au contrôleur

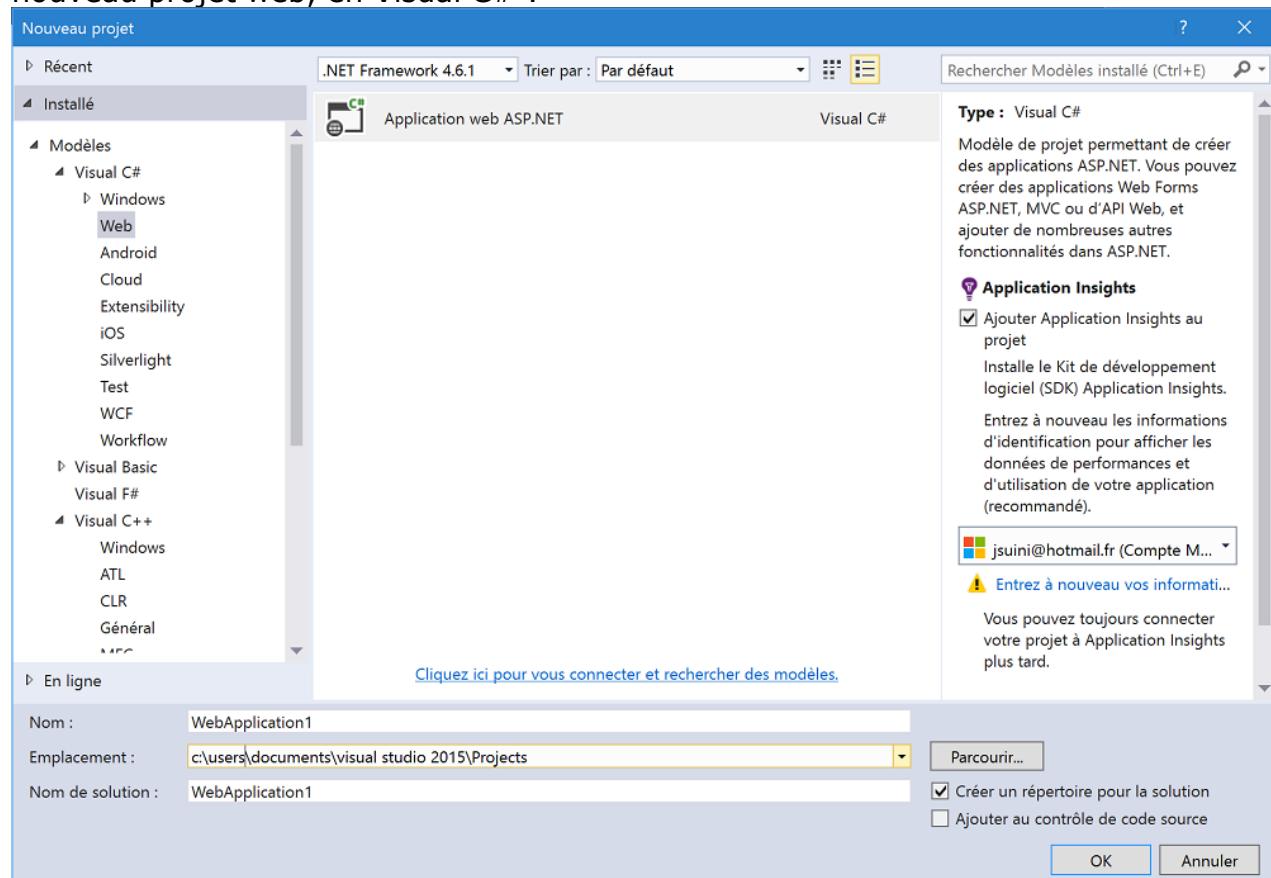
4 - Le contrôleur décide de la vue à afficher, va l'appeler

5 - Le code HTML de la vue est envoyé à l'utilisateur pour qu'il puisse naviguer normalement

### Création d'une application ASP.NET MVC sous Visual Studio.

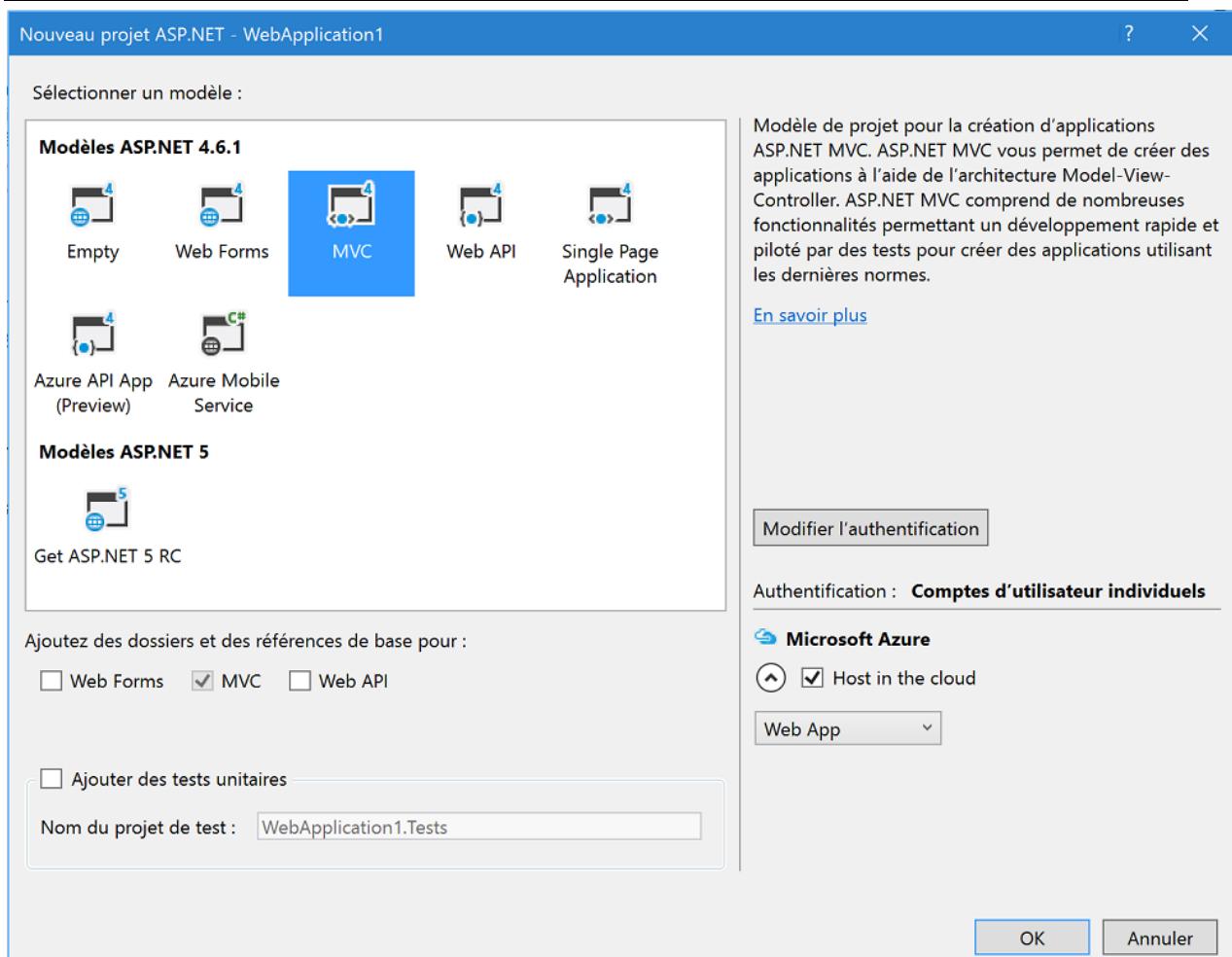
#### 1. Création d'un site

Pour créer une application web ASP.NET MVC, nous commençons par créer un nouveau projet web, en Visual C# :

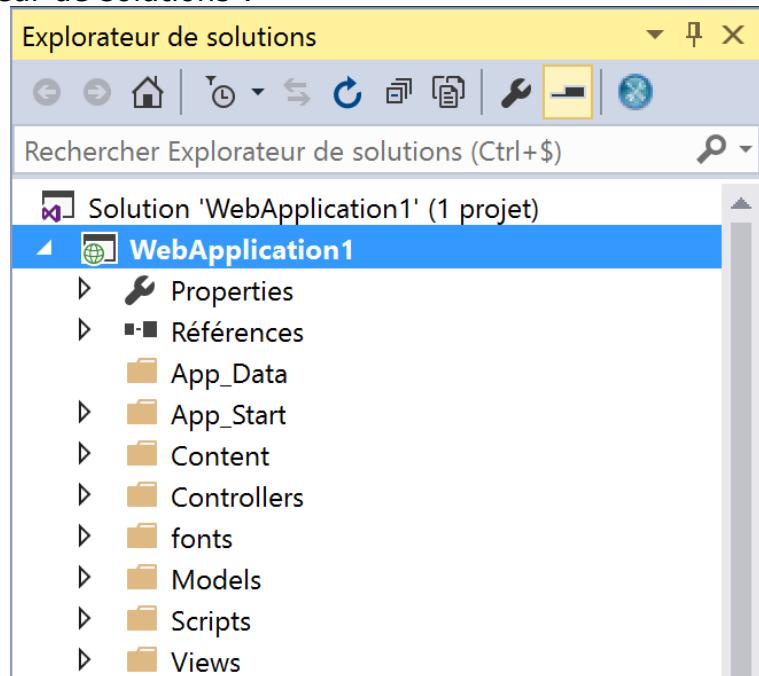


Après avoir entré le nom de notre projet et défini son emplacement, nous sélectionnons le modèle MVC en conservant les options proposées par défaut :

## MVC ASP.NET



Notre projet (ici WebApplication1) est maintenant créé, jetons un coup d'œil à notre explorateur de solutions :



Concernant les différents éléments présentés dans la liste hiérarchique :

App_Start	Contient les instructions de configuration exécutées au démarrage de l'application
Content	Contient les feuilles de style CSS
Controllers	Contient nos contrôleurs ayant pour but de traiter nos actions
Fonts	Les polices de caractères téléchargées par le navigateur
Models	Regroupe nos modèles, destinés à regrouper les données
Scripts	Regroupe des modules de type JavaScript, jQuery et AJAX
Views	Regroupe l'ensemble de nos vues

Par défaut, les projets MVC incluent les dossiers suivants :

- App\_Data, qui est la base physique des données. Ce dossier a le même rôle que dans les sites Web ASP.NET qui utilisent des pages Web Forms.
- Content, qui est l'emplacement recommandé pour l'ajout des fichiers de contenu tels que les fichiers de feuille de style en cascade, les images, etc. En général, le dossier Content est réservé aux fichiers statiques.
- Controllers, qui est l'emplacement recommandé pour les contrôleurs. L'infrastructure MVC requiert que les noms de tous les contrôleurs se terminent par « Controller », tels que HomeController, LoginController ou ProductController.
- Models, qui est fourni pour les classes représentant le modèle d'application de votre application Web MVC. Ce dossier inclut généralement du code définissant les objets et la logique utilisée pour l'interaction avec le magasin de données. En général, les objets de modèle réels sont placés dans des bibliothèques de classes séparées. Toutefois, lorsque vous créez une application, vous pouvez placer les classes dans ce dossier, puis les déplacer vers des bibliothèques de classes séparées ultérieurement dans le cycle de développement.
- Scripts, qui est l'emplacement recommandé pour les fichiers de script prenant en charge l'application. Par défaut, ce dossier contient les fichiers de base ASP.NET AJAX et la bibliothèque jQuery.
- Views, qui est l'emplacement recommandé pour les vues. Les vues utilisent les fichiers ViewPage (.aspx), ViewUserControl (.ascx) et ViewMasterPage (.master), en plus des autres fichiers liés au rendu des vues. Le dossier Views contient un dossier pour chaque contrôleur ; chaque dossier est nommé avec le préfixe du nom de contrôleur. Par exemple, si un contrôleur est nommé HomeController, le dossier Views contient un dossier appelé

Home. Par défaut, lorsque l'infrastructure ASP.NET MVC charge une vue, elle recherche un fichier ViewPage (.aspx) portant le nom de vue requis dans le dossier Views\NomContrôleur. Par défaut, un dossier nommé Shared, qui ne correspond à aucun contrôleur, est également présent dans le dossier Views. Le dossier Shared est utilisé pour les vues partagées par plusieurs contrôleurs. Par exemple, vous pouvez placer la page maître de l'application Web dans le dossier Shared.

Outre les dossiers indiqués précédemment, une application Web MVC utilise le code du fichier Global.asax pour définir des valeurs par défaut de routage d'URL globales et fait appel au fichier Web.config pour configurer l'application.

### 2. Organisation des répertoires

La solution du projet web contient beaucoup plus de répertoires qu'un projet Web Forms. Ces répertoires ont pour but de guider le programmeur :

App_Start	Instructions de configuration exécutées au démarrage du site.
Content	Contient les feuilles de style CSS et autres ressources partagées.
Controllers	Regroupe les contrôleurs destinés à traiter les actions.
fonts	Polices de caractères téléchargées par le navigateur.
Models	Rassemble les modèles qui sont des entités métier.
Scripts	Ensemble de modules JavaScript, jQuery et AJAX.
Views	Vue .cshtml.

Les vues sont des pages web mais qui n'ont pas de code sous-jacent (voir ci-dessous). Elles sont en principe regroupées dans des dossiers appelés zones, lesquels correspondent à des contrôleurs. Cette règle n'a aucun caractère obligatoire d'un point de vue technique, même si l'emploi du framework s'en trouve facilité dans le cas où elle est appliquée.

### 3. Création du modèle

Un modèle est une classe dont les instances sont appelées "objets métier". Cela signifie qu'aucune logique technique n'y figure, et que le framework est chargé de gérer le cycle de vie du composant métier, et de lui apporter des services techniques de haut niveau tels que la sécurité, les transactions, la validation, la persistance...

```
#region Modèle Ouvrage
/// <summary>
/// Objet métier Ouvrage
/// </summary>
```

```
public class Ouvrage
{
    /// <summary>
    /// Par convention, ID a une clé primaire
    /// </summary>
    public int ID { get; set; }

    /// <summary>
    /// Champ Auteur
    /// </summary>
    public string Auteur { get; set; }

    /// <summary>
    /// Champ Titre
    /// </summary>
    public string Titre { get; set; }

    public Ouvrage()
    {
        ID = 0;
        Auteur = "";
        Titre = "";
    }

    public Ouvrage(int id, string auteur, string titre)
    {
        ID = id;
        Auteur = auteur;
        Titre = titre;
    }
}

#endregion
```

Nous avons associé une classe de service qui contient quelques méthodes incontournables. Les classes de service implémentent assez souvent le pattern CRUD (Create Read Update Delete). Nous ne nous écartons pas de cette pratique, sans toutefois implémenter la persistance en base SQL, afin de nous concentrer sur l'architecture MVC.

```
#region OuvrageService
```

```
/// <summary>
/// Classe de service pour Ouvrage
/// </summary>
public class OuvrageService
{
    /// <summary>
    /// Fournit une liste d'ouvrages
    /// </summary>
    /// <returns></returns>
    public List<Ouvrage> Select()
    {
        List<Ouvrage> l = new List<Ouvrage>();
        l.Add(new Ouvrage(1, "Brice-Arnaud", "ASP.NET 4.5.2 avec C#"));
        l.Add(new Ouvrage(2, "Brice-Arnaud", "Conduite de projets"));
        l.Add(new Ouvrage(3, "Brice-Arnaud", "Langage C++"));

        return l;
    }

    /// <summary>
    /// Recherche un ouvrage à partir de sa clé primaire
    /// </summary>
    /// <param name="id">Clé primaire</param>
    /// <returns></returns>
    public Ouvrage Select(int id)
    {
        // utilise LINQ pour effectuer la recherche
        // une boucle for conviendrait aussi
        var q = from o in Select() where o.ID == id select o;
        return q.ToArray()[0];
    }

    public void Insert(Ouvrage ouvrage)
    {
        // ...
    }

    public void Update(Ouvrage ouvrage)
```

```
{  
    // ...  
}  
  
public void Delete(int id)  
{  
    // ...  
}  
}  
#endregion
```

#### 4. Définition du contrôleur

Un contrôleur est une classe qui dérive de Controller. Utilisez l'assistant pour ajouter un contrôleur au projet : Dans ce premier exemple, choisissez un modèle de contrôleur sans code préexistant pour créer la classe OuvragesController. La classe contrôleur est activée par le biais d'une URL qui est configurée au moyen de routes. Ces URL sont rappelées sous forme de commentaires dans le code qui suit :

```
public class OuvragesController : Controller  
{  
    // URL d'accès :  
    // GET: /Ouvrages/  
    /// <summary>  
    /// Action Index.  
    /// Conduit à la vue Index chargée d'afficher la liste  
    /// des ouvrages  
    /// </summary>  
    /// <returns>Vue courante</returns>  
    public ActionResult Index()  
    {  
        var p = new OuvrageService();  
  
        // modèle  
        var ouvrages = p.Select();  
  
        // la vue doit rendre le modèle ouvrages  
        return View(ouvrages);  
    }  
}
```

```
// URL d'accès :  
// GET: /Ouvrages/Selection/3  
/// <summary>  
/// Action Selection.  
/// Conduit à la vue Selection chargée d'afficher le détail  
/// d'un ouvrage  
/// </summary>  
/// <param name="index">clé primaire</param>  
/// <returns></returns>  
public ActionResult Selection(int? id)  
{  
    var p = new OuvrageService();  
  
    // modèle correspondant à la recherche  
    var ouvrage = p.Select(id.HasValue ? id.Value : 1);  
  
    // la vue doit rendre le modèle ouvrage  
    return View(ouvrage);  
}  
}
```

### 5. Ajout des vues

Visual Studio propose un assistant pour créer de nouvelles vues à partir des informations fournies par un modèle et par les indications du programmeur. Seul le nom de la vue est obligatoire, mais cet écran propose des options importantes, comme le modèle sous-jacent (*Ouvrage*) et la composition de la page (*List*).

L'implémentation de la vue *Index* est assez courte :

```
@model IEnumerable<chapitre4_mvc_demo.Models.Ouvrage>  
  
{@  
    Layout = null;  
}  
  
<!DOCTYPE html>
```

```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <p>
        @Html.ActionLink("Ajouter", "Create")
    </p>
    <table class="table">
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Auteur)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Titre)
            </th>
            <th></th>
        </tr>

        @foreach (var item in Model) {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Auteur)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Titre)
                </td>
                <td>
                    @Html.ActionLink("Modifier", "Edit", new { id=item.ID }) |
                    @Html.ActionLink("Détails", "Selection", new { id=item.ID }) |
                    @Html.ActionLink("Supprimer", "Delete", new { id=item.ID })
                </td>
            </tr>
        }
    </table>
</body>
```

```
</html>
```

Dans notre cas, la classe Vue hérite du Modèle. Cette approche diffère singulièrement des pratiques Java, mais reste au contraire identique à l'architecture Web Form. On remarque également que la classe de base est un type générique paramétré pour un modèle particulier :

```
System.Web.Mvc.ViewPage<IEnumerable<chapitre4_mvc.Models.Ouvrage>>
```

S'agissant d'une liste, la vue réalise une itération à l'aide d'un scriptlet for. Nous en déduisons l'existence d'une variable Model de type **IEnumerable<Ouvrage>** :

```
<% foreach (var item in Model) { %>
```

L'objet **Html** expose la méthode **ActionLink** qui produit la valeur d'un lien correspondant à l'action Selection. Le troisième paramètre est la valeur de l'attribut **ID** servant de clé primaire. Le premier paramètre est tout simplement le texte du lien :

```
<%: Html.ActionLink("Details", "Selection", new { id=item.ID })%> |
```

À l'exécution, nous constatons que le lien généré correspond bien à la forme de l'URL décrite ci-dessus dans la classe contrôleur.

### Définition des routes

Le fichier **RoutesConfig** contient la définition des routes ; une route est une URL qui associe un contrôleur, une action et des paramètres optionnels :

```
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 }
        );
    }
}
```

```
}
```

Lorsque le framework cherche une route, la liste est parcourue de haut en bas, dans l'ordre, et la localisation s'arrête dès qu'une correspondance est trouvée. Ainsi, la route par défaut doit-elle toujours figurer en dernier.

Dans notre cas, comme le paramètre ID est optionnel, l'action Selection doit recevoir un type nullable int?. Ainsi dans l'exemple suivant, ce paramètre est omis et l'implémentation de l'action présume qu'IDa la valeur implicite 1.

### 1. D'une action à l'autre

Nous faisons évoluer notre exemple de librairie par l'ajout d'un modèle de commande :

```
public class Commande
{
    public int IDCommande { get; set; }
    public Ouvrage ouvrage { get; set; }
    public int Quantite { get; set; }
    public string Email { get; set; }

    public Commande()
    {
        Quantite = 1;
        IDCommande = 0;
        ouvrage = new Ouvrage();
        Email = "";
    }

    public Commande(int idCommande,Ouvrage ouvrage,int
quantite,string email)
    {
        this.IDCommande = IDCommande;
        this.ouvrage = ouvrage;
        this.Quantite = quantite;
        this.Email = email;
    }
}
```

## MVC ASP.NET

Une classe de service supporte les opérations de base et utilise la session HTTP plutôt que SQL comme moyen de persistance.

```
public class CommandesServices
{
    #region Commandes
    public List<Commande> Commandes
    {
        get
        {
            List<Commande> l = null;
            l =
(List<Commande>) HttpContext.Current.Session["Commandes"];
            if (l == null)
            {
                l = new List<Commande>();
                HttpContext.Current.Session["Commandes"] = l;
            }

            return l;
        }
    }
    #endregion

    #region IDCommande
    public int IDCommande
    {
        get
        {
            int? id = null;
            if (HttpContext.Current.Session["IDC"] == null)
            {
                id = 1;
                HttpContext.Current.Session["IDC"] = id;
            }

            return id.Value;
        }
    }
}
```

```
set
{
    HttpContext.Current.Session["IDC"] = value;
}
}

#endregion

#region CreerCommande
public void CreerCommande(Commande commande)
{
    commande.IDCommande = IDCommande++;
    Commandes.Add(commande);
}
#endregion

#region RetrouverCommande
public Commande RetrouverCommande(int idCommande)
{
    var q = from c in Commandes where c.IDCommande ==
idCommande select c;
    return q.ToArray()[0];
}
#endregion

#region ModifierCommande
public void ModifierCommande(Commande commande)
{
    var q = from c in Commandes where c.IDCommande ==
commande.IDCommande select c;
    var cl = q.ToArray()[0];

    cl.ouvrage.Auteur = commande.ouvrage.Auteur;
    cl.ouvrage.Titre = commande.ouvrage.Titre;
    cl.ouvrage.ID = commande.ouvrage.ID;

    cl.Email = commande.Email;
    cl.Quantite = commande.Quantite;
}
```

```
}
```

```
#endregion
```

```
}
```

Vient s'ajouter également une vue Commande/Create pour que l'utilisateur puisse saisir sa commande d'ouvrage :

```
@model chapitre4_mvc_demo.Models.Commande
```

```
@{
```

```
    Layout = null;
```

```
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <meta name="viewport" content="width=device-width" />
```

```
    <title>Create</title>
```

```
</head>
```

```
<body>
```

```
    @Scripts.Render("~/bundles/jquery")
```

```
    @Scripts.Render("~/bundles/jqueryval")
```

```
    @using (Html.BeginForm())
```

```
    {
```

```
        @Html.AntiForgeryToken()
```

```
            <div class="form-horizontal">
```

```
                <h4>Contenu de la commande</h4>
```

```
                <hr />
```

```
                @Html.ValidationSummary(true)
```

```
                <div class="form-group">
```

```
                    @Html.LabelFor(model => model.IDCommande, new {
```

```
                        @class = "control-label col-md-2" })
```

```
                    <div class="col-md-10">
```

```
                        @Html.EditorFor(model => model.IDCommande)
```

```
    @Html.ValidationMessageFor(model =>
model.IDCommande)
    </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Quantite, new
{ @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Quantite)
            @Html.ValidationMessageFor(model =>
model.Quantite)
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Email, new { @class
= "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Email)
            @Html.ValidationMessageFor(model =>
model.Email)
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.ouvrage.Auteur, new
{ @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model =>
model.ouvrage.Auteur)
            @Html.ValidationMessageFor(model =>
model.ouvrage.Auteur)
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.ouvrage.Titre, new
```

```
{ @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model =>
model.ouvrage.Titre)
        @Html.ValidationMessageFor(model =>
model.ouvrage.Titre)
    </div>
</div>

<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <input type="submit" value="Create"
class="btn btn-default" />
    </div>
</div>
</div>

<div>
    @Html.ActionLink("Retour à la liste des commandes",
"Index")
</div>
</body>
</html>
```

Pour rendre cette vue opérationnelle, il faut l'appeler depuis celle qui détaille un ouvrage, et passer à l'action un paramètre identifiant l'ouvrage.

Pour cela, nous avons besoin d'un contrôleur :

```
public class CommandesController : Controller
{
    //
    // GET: /Commandes/Create/5

    public ActionResult Create(int? IdOuvrage)
    {
        var c = new Commande();
        var ouvrage = new
OuvrageService().Select(idOuvrage.HasValue?idOuvrage.Value:1);
```

```
c.ouvrage = ouvrage;

return View(c);
}

}
```

Dans la vue Selection, nous ajoutons un lien destiné à atteindre l'action Create du contrôleur Commandes. On remarquera comment le paramètre idOuvrage est identifié et valorisé :

```
<tr>
    <td></td>
    <td>@Html.ActionLink("Commander", "Create", "Commandes", new
    { idOuvrage = Model.ID }, null)</td>
</tr>
```

À l'exécution, nous vérifions bien que ce paramètre est passé dans l'URL et que l'ouvrage sélectionné est correctement référencé par la commande :

## 2. Mise à jour du modèle et redirection

Le contrôleur CommandesController reçoit une deuxième définition de l'action Create. Celle-ci réagit sur un POST HTTP et a pour but d'enregistrer une nouvelle commande, puis de router le navigateur sur la vue Index :

```
//
// POST: /Commandes/Create
[HttpPost]
public ActionResult Create(int? idOuvrage, Commande post)
{
    var p = new CommandesServices();
    var c = new Commande();

    // met à jour la commande à partir du formulaire
    UpdateModel(c);

    var ouvrage = new OuvrageService().Select(idOuvrage.HasValue
        ? idOuvrage.Value : 1);
    c.ouvrage = ouvrage;

    p.CreerCommande(c);
```

```
return RedirectToAction("Index");  
}
```

L'action et la vue Index sont chargées de lister les commandes, leur conception est donc identique à ce qui a été présenté précédemment pour les ouvrages.

### 3. Validation

Comme la validation des données métiers n'a rien de technique, elle peut s'effectuer par le biais d'annotations, c'est-à-dire d'attributs spécifiés au niveau des propriétés du modèle.

```
public class Commande  
{  
    [DisplayName("N°")]  
    public int IDCommande { get; set; }  
  
    public Ouvrage ouvrage { get; set; }  
  
    [Range(1, 100, ErrorMessage = "La quantité doit être comprise  
entre 1 et 10")]  
    public int Quantite { get; set; }  
  
    [Required(ErrorMessage = "Indiquez l'adresse Email")]  
    [DataType(DataType.EmailAddress, ErrorMessage = "Format  
d'email attendu")]  
    public string Email { get; set; }  
}
```

L'action Create du contrôleur n'a plus qu'à vérifier si le modèle est valide ; dans l'affirmative, il peut procéder à l'enregistrement de la commande, sinon, il faut renvoyer l'utilisateur sur l'écran de saisie pour qu'il soit avisé de la non-conformité :

```
[HttpPost]  
public ActionResult Create(int? idOuvrage, Commande post)  
{  
    if (!ModelState.IsValid)  
    {  
        ViewData["message"] = "La commande n'est pas valide";  
        return View();  
    }  
}
```

```
var p = new CommandesServices();

var c = new Commande();

// met à jour la commande à partir du formulaire
UpdateModel(c);

var ouvrage = newOuvrageService().Select(idOuvrage.HasValue
? idOuvrage.Value : 1);

c.ouvrage = ouvrage;

p.CreerCommande(c);

return RedirectToAction("Index");

}
```

L'objet Html dispose, lui, d'une méthode ValidationSummary chargée de lister l'ensemble des erreurs remontées par le système de validation intégré au framework :

```
<body>
<% using (Html.BeginForm()) {%
    <%: Html.ValidationSummary(true, "Erreurs") %>
```

### Le moteur de rendu Razor et les vues

Comme alternative aux Web Forms .aspx, Microsoft propose Razor, système de rendu de pages HTML. Celui-ci a comme caractéristique d'alléger et d'optimiser le rendu tout en simplifiant la syntaxe utilisée pour décrire des pages. Razor est accessible en particulier aux sites MVC qui ne disposent pas de composants riches comme les Web Forms mais qui en contrepartie économisent l'emploi d'un champ caché ViewState très volumineux.

#### 1. La syntaxe C# dans les vues CSHTML

##### a. Principes de base

Le moteur Razor intègre finement le langage C# (tout comme VB.NET) dans les vues et a comme base de sa syntaxe une simplification des balises scriptlets. Ce mélange de syntaxe C# et de code HTML est désigné par l'extension de pages .cshtml.

En étudiant le gabarit de vues \_Layout.cshtml situé dans le répertoire Shared, on peut observer que le signe @ sert de marqueur au moteur pour distinguer les séquences C# des balises HTML.

```
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - BAG</p>
    </footer>
</div>
```

Cette syntaxe rend inutiles les marqueurs `<% %>` et `<%= %>` car le moteur reconnaît directement dans les pages des instructions C# comme `@RenderBody()` ou `@DateTime.Now`.

Il existe une façon d'échapper le signe @ en le doublant, ce qui est utile pour écrire directement une adresse e-mail.

Voici maintenant l'exemple de l'action Index appartenant au contrôle Home.

```
public ActionResult Index()
{
    // instanciation du modèle
    HomeModel m = new HomeModel();

    // définition de données
    m.Themes = new List<string>();
    m.Themes.Add("ASP.NET");
    m.Themes.Add("Conduite de projets");
    m.Themes.Add("C++");

    // utilisation d'une propriété dynamique
    ViewBag.Categorie = "Informatique";

    // la vue Index est associée au modèle m
    return View(m);
}
```

La vue correspondante Index annonce l'utilisation du modèle HomeModel. Elle affiche les données qu'il contient ainsi que la propriété dynamique définie dans le contrôleur. La syntaxe pour introduire du code C# dans la séquence HTML est on ne peut plus simple et directe, beaucoup plus lisible que l'équivalent en Web Form :

```
@model chapitre4_mvc_demo.Models.HomeModel
{
    ViewBag.Title = "Accueil";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>www.malibrairieasp.com</h2>
<h3>
    Rechercher des ouvrages
</h3>
<div>
    <span>Catégorie : @ViewBag.Categorie</span>
</div>
<div>
    <ul>
        @foreach (var t in Model.Themes)
        {
            <li>@t</li>
        }
    </ul>
</div>
```

### b. Les balises Action

Microsoft a défini une classe Html qui expose des méthodes de génération de script. Avant de voir comment étendre cette classe, voici son utilisation dans l'appel d'actions de contrôleurs.

La méthode Action sert à rendre l'exécution d'une action dans une vue (généralement partielle). Dans notre vue Index, nous appelons l'action Publicite :

```
<div>
    @Html.Action("Publicite")
</div>
```

Dans le contrôleur, l'action Publicite va retourner une vue partielle du même nom. On peut optionnellement employer un modèle ou une propriété du ViewBag comme dans notre exemple :

```
public ActionResult Publicite()
{
    ViewBag.Promo = "Les livres ASP.NET";
    return PartialView();
}
```

La vue partielle Publicite affiche la propriété du ViewBag définie dans le contrôleur :

```
<h3>
    Promotions : @ViewBag.Promo
</h3>
```

À l'exécution, nous vérifions que le résultat de l'action s'insère dans la vue Index :

Nous avons également rencontré lors de précédents exemples la méthode ActionLink qui génère un tag de type lien hypertexte :

```
<div>
    @Html.ActionLink("Retour à la liste", "Index")
</div>
```

Enfin, @Url.Action retourne l'URL d'une action, avec ses paramètres, ce qui peut être utile lorsque l'on doit personnaliser le code JavaScript ou employer une autre balise qu'un lien hypertexte :

```
<script>
    function goOuvrages() {
        window.location='@Url.Action("Index", "Ouvrages")'
    }

</script>
<div>
    <input type="button" onclick="goOuvrages()" value="Afficher
    les ouvrages">
</div>
```

### c. Les méthodes de formulaires

Les méthodes de formulaires servent à mettre en correspondance les balises HTML <input> avec les propriétés d'un modèle :

@Html.AntiForgeryToken	Génère un code de sécurité pour éviter les attaques.
@Html.BeginForm	Ouvre la balise <form>. Voir aussi EndForm.
@Html.Checkbox	Produit une case à cocher pour un champ.
@Html.Editor	Produit une zone de saisie pour un champ.
@Html.Label	Produit un label pour un champ.
@Html.ListBox	Produit une liste pour un champ.
@Html.Password	Produit une zone de saisie de mot de passe.
@Html.RadioButton	Produit un bouton radio.
@Html.TextArea	Produit une zone de texte multiligne.
@Html.TextBox	Produit une zone de texte.

Ces méthodes sont complétées d'une variante For qui a été utilisée au début de ce chapitre. La variante For présente l'intérêt d'indiquer directement la propriété du modèle, ce qui limite les risques d'erreurs en comparaison à une mention "en dur" dans une chaîne de caractères.

```
@Html.TextBoxFor(model => model.Titre, new { @readonly =
"readonly" })
```

La liste des types de contrôles de formulaire est somme toute assez courte et elle traduit bien l'orientation du framework MVC avec Razor : les terminaux mobiles ne sont pas conçus pour des interfaces complexes et l'optimisation du rendu est d'autant plus facile que les séquences sont courtes.

### d. Créer ses propres extensions HTML

La technologie MVC désigne ces extensions sous le nom helper (facilitateur). Il existe deux moyens de créer ces extensions : soit par une méthode statique, soit par une extension de la classe HtmlHelper.

La classe suivante présente les deux approches :

```
public static class MyHelpers
{
    /// <summary>
    /// Helper HTML par méthode statique
    /// </summary>
```

```
public static string Aujourd'hui(string cultureInfo)
{
    string res = DateTime.Now.ToString(new
CultureInfo(cultureInfo));
    return res;
}

/// <summary>
/// Helper HTML par extension de la classe HtmlHelpers
/// </summary>
public static string Aujourd'hui(this HtmlHelper
helper, string cultureInfo)
{
    string res = DateTime.Now.ToString(new
CultureInfo(cultureInfo));
    return res;
}
}
```

À l'utilisation, seul le préfixe distingue ces deux versions qui donnent évidemment le même résultat :

```
<h2>Définir des helpers HTML</h2>
<div>
    <span>Avec une classe statique : </span>
    @MyHelpers.Aujourd'hui("fr-fr")
</div>
<div>
    <span>Avec une extension de classe : </span>
    @Html.Aujourd'hui("fr-fr")
</div>
```

## 2. Structure et organisation des vues

L'approche MVC pousse assez loin la modularité, toujours dans un esprit de simplicité et de maintenance aisée du site. Les vues sont une bonne illustration de ce principe.

### a. Les gabarits Layout

Les gabarits (aussi appelés layout) ont le même rôle que les pages maîtres (master pages) des Web Forms. Par convention, les layouts sont placés dans le répertoire Shared et commencent par un caractère souligné, bien qu'aucune contrainte technique ne vienne s'imposer au développeur. Le gabarit principal \_Layout défini par Visual Studio au moment de la création du site vient installer et structurer l'ensemble des pages.

Il faut prêter attention dans le code qui suit aux instructions de génération des balises de style, de script, et du rendu principal de la vue :

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>@ViewBag.Title - www.malibrairieasp.com</title>
@Styles.Render("~/Content/css")
@Scripts.Render("~/bundles/modernizr")

</head>
<body>
<div class="navbar navbar-inverse navbar-fixed-top">
<div class="container">
<div class="navbar-header">
<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
@Html.ActionLink("www.malibrairieasp.com", "Index",
"Home", null, new { @class = "navbar-brand" })
</div>
<div class="navbar-collapse collapse">
<ul class="nav navbar-nav">
<li>@Html.ActionLink("Accueil", "Index",
"Home")</li>
```

```
<li>@Html.ActionLink("Ouvrages", "Index",
    "Ouvrages")</li>
    <li>@Html.ActionLink("Contact", "Contact",
    "Home")</li>
</ul>
@Html.Partial("_LoginPartial")
</div>
</div>
</div>
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - BAG</p>
    </footer>
</div>

@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)
</body>
</html>
```

Lorsque l'on définit une vue, il n'est pas utile de reprendre le contenu du gabarit `_Layout`, mais en revanche il faut s'y référer :

```
@{
    ViewBag.Title = "Aide";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

### b. Les vues partielles

Les vues partielles servent à rendre modulaires des vues qui s'emboîtent. Les syntaxes `RenderPartial(vue<,modèle>)` et `Partial(vue<,modèle>)` récupèrent le contenu d'une vue partielle à insérer dans la vue principale. La méthode `Partial()` retourne une chaîne qui peut être stockée dans une variable alors que `RenderPartial()` appelle la méthode interne `Write`. Dans certains cas, on peut employer l'une ou l'autre pour obtenir le même résultat.

```
<footer>
```

```
@Html.Partial("_Pied")  
</footer>
```

La vue partielle - qui peut exploiter un modèle - a exactement le fonctionnement qu'une autre vue. Dans notre exemple, elle sert à partager du code HTML statique :

```
<div>  
    &copy; 2015 BAG  
</div>
```

### c. Rendu des scripts et des bundles

Désormais, les scripts fonctionnent par bibliothèque et non plus par fichier isolé. Pour simplifier leur utilisation, les bundles les regroupent par thème ou par fonction.

La définition des bundles s'effectue dans la classe BundleConfig appelée au démarrage de l'application par le fichier startup.cs :

```
public class BundleConfig  
{  
    public static void RegisterBundles(BundleCollection bundles)  
    {  
        bundles.Add(new ScriptBundle("~/bundles/jquery").Include(  
            "~/Scripts/jquery-{version}.js");  
  
        bundles.Add(new ScriptBundle("~/bundles/jqueryval").  
            Include("~/Scripts/jquery.validate*"));  
    }  
}
```

Ensuite, dans les vues - souvent les gabarits Layout - on invoque le rendu des bundles, ce qui évite d'avoir à déclarer les fichiers de script un par un :

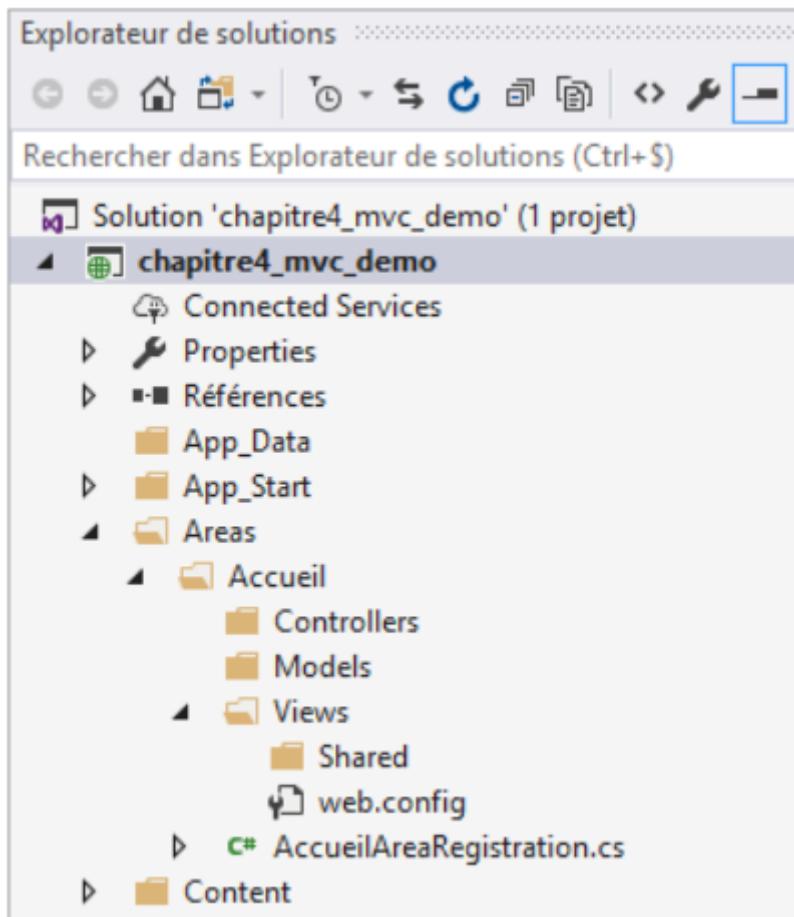
```
@Scripts.Render("~/bundles/jquery")  
@Scripts.Render("~/bundles/jqueryval")
```

### Définir des zones (areas)

Les zones servent à regrouper dans des répertoires tous les éléments relatifs à un "module" fonctionnel, c'est-à-dire les contrôleurs, les vues et les modèles. La commande Ajouter - Zone est accessible depuis l'Explorateur de solutions :

## MVC ASP.NET

Après avoir indiqué le nom de la zone ("Accueil" dans notre exemple), Visual Studio initialise la zone en créant des répertoires ainsi qu'un fichier de configuration de routes :



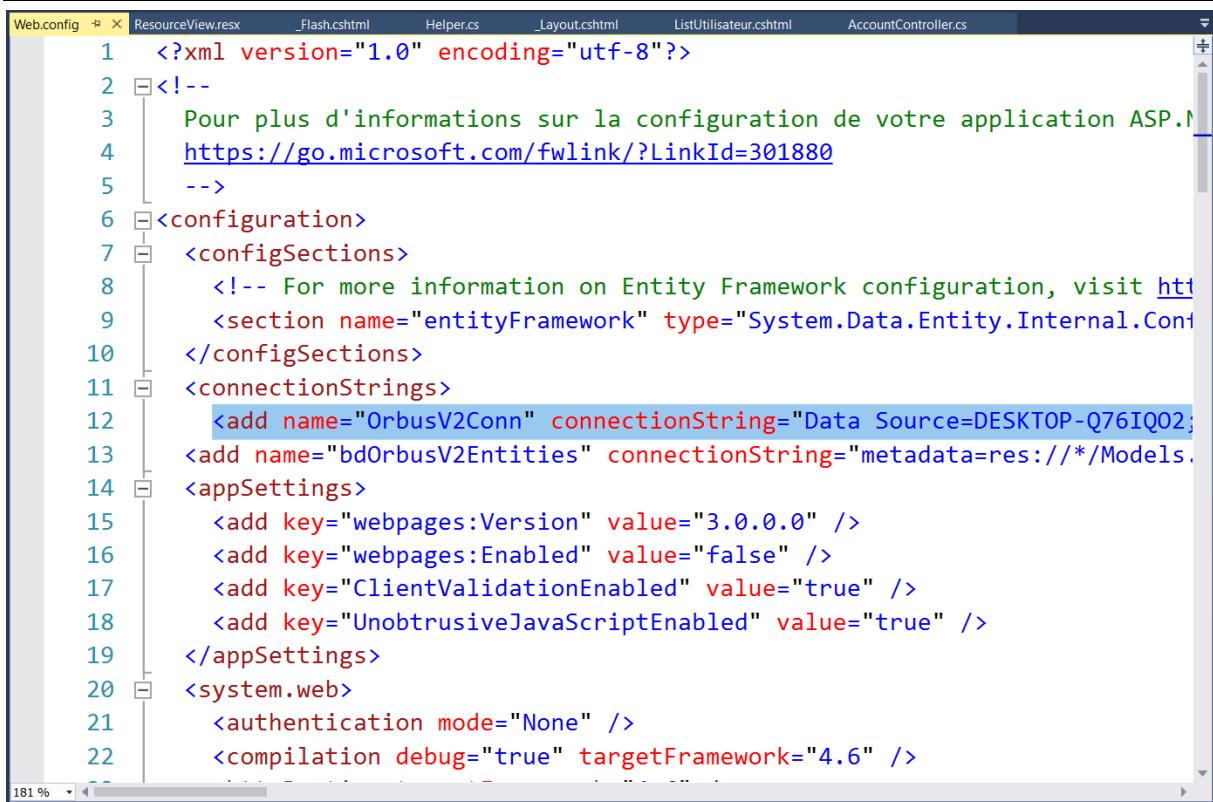
Le développement du site n'est en rien modifié, seule l'organisation des fichiers et des routes (URL) est un peu différente. En effet, le nom de la zone vient s'intercaler dans l'URL :

Mise en route du CodeFirst.

Chaine de connexion

```
<add name="OrbusV2Conn" connectionString="Data Source=DESKTOP-Q76IQ02;Initial Catalog=bdOrbusV2;Integrated Security=True" providerName="System.Data.SqlClient" />
```

## MVC ASP.NET



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!--
3      Pour plus d'informations sur la configuration de votre application ASP.NET,
4      visitez https://go.microsoft.com/fwlink/?LinkId=301880
5  -->
6  <configuration>
7      <configSections>
8          <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237493-->
9          <section name="entityFramework" type="System.Data.Entity.Internal.Configurations.EntityFrameworkSection, System.Data.Entity, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
10     </configSections>
11     <connectionStrings>
12         <add name="OrbusV2Conn" connectionString="Data Source=DESKTOP-Q76IQ02;Initial Catalog=OrbusV2;Integrated Security=True" />
13         <add name="bdOrbusV2Entities" connectionString="metadata=res://*/Models.OrbusV2.csdl|res://*/Models.OrbusV2.ssdl|res://*/Models.OrbusV2.msl;provider=System.Data.SqlClient;provider connection string='Data Source=DESKTOP-Q76IQ02;Initial Catalog=bdOrbusV2;Integrated Security=True'" />
14     </connectionStrings>
15     <appSettings>
16         <add key="webpages:Version" value="3.0.0.0" />
17         <add key="webpages:Enabled" value="false" />
18         <add key="ClientValidationEnabled" value="true" />
19         <add key="UnobtrusiveJavaScriptEnabled" value="true" />
20     </appSettings>
21     <system.web>
22         <authentication mode="None" />
23         <compilation debug="true" targetFramework="4.6" />
24     </system.web>
25 
```

### Context

```
public class OrbusV2Context : DbContext
{
    public OrbusV2Context() : base("OrbusV2Conn")
    {
    }
    public DbSet<Utilisateur> Utilisateurs { get; set; }
    public DbSet<T_ForgetPassword> T_ForgetPassword { get; set; }
    public DbSet<HisChangePwd> HisChangePwd { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        //The object and class have the same name
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}
```

## MVC ASP.NET

The screenshot shows the Microsoft Visual Studio interface with the title bar "MVC ASP.NET". The menu bar includes Fichier, Edition, Affichage, Projet, Générer, Déboguer, Équipe, Outils, Test, Analyser, Fenêtre, Aide. The toolbar has icons for Save, Undo, Redo, Cut, Copy, Paste, Find, etc. The status bar shows "181 %". The code editor window displays the file "OrbusV2Context.cs" with the following content:

```
4  using System.Web;
5  using System.Data.Entity;
6  using System.Data.Entity.ModelConfiguration.Conventions;
7
8
9  namespace MvcOrbusV2.Models
10 {
11     public class OrbusV2Context : DbContext
12     {
13         public OrbusV2Context() : base("OrbusV2Conn")
14         {
15         }
16         public DbSet<Utilisateur> Utilisateurs { get; set; }
17         public DbSet<T_ForgetPassword> T_ForgetPassword { get; set; }
18         public DbSet<HisChangePwd> HisChangePwd { get; set; }

21     protected override void OnModelCreating(DbModelBuilder modelBuilder)
22     {
23         //The objet and class have the same name
24         modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
25     }
}
```

### CodeFirst

The screenshot shows the NuGet Package Manager Console with the following text:

Console du Gestionnaire de package  
Source de packages : Tout | Projets par défaut : MvcOrbusV2

et n'octroie aucune licence les concernant. Certains packages peuvent nécessiter des licences supplémentaires. Suivez l'URL (flux) de la source de package pour plus d'informations.

Version de l'hôte de la Console du Gestionnaire de package 4.5.0.0

Tapez 'get-help NuGet' pour afficher toutes les commandes NuGet disponibles.

```
PM> Enable-Migrations -ContextTypeName OrbusV2Context
```

## MVC ASP.NET

The screenshot shows the Visual Studio IDE with the code editor open. The file being edited is `MvcOrbusV2\Migrations\Configuration.cs`. The code defines a sealed class `Configuration` that inherits from `DbMigrationsConfiguration<MvcOrbusV2Context>`. It contains a constructor that sets `AutomaticMigrationsEnabled` to `true` and `AutomaticMigrationDataLossAllowed` to `false`. A protected override method `Seed` is defined, which includes comments about calling it after migration and avoiding duplicate seed data using `AddOrUpdate`.

```
4     using System.Data.Entity;
5     using System.Data.Entity.Migrations;
6     using System.Linq;
7
8     internal sealed class Configuration : DbMigrationsConfiguration<MvcOrbusV2Context>
9     {
10        public Configuration()
11        {
12            AutomaticMigrationsEnabled = true;
13            AutomaticMigrationDataLossAllowed = false;
14        }
15
16        protected override void Seed(MvcOrbusV2.Models.OrbusV2Context context)
17        {
18            // This method will be called after migrating to the latest version.
19
20            // You can use the DbSet<T>.AddOrUpdate() helper extension method
21            // to avoid creating duplicate seed data. E.g.
22            //
23            //    context.People.AddOrUpdate(
24            //        p => p.FullName,
25            //        new Person { FullName = "Andrew Peters" } );

```

### Mise à Jour de la base

The screenshot shows the NuGet Package Manager Console window. The title bar says "Console du Gestionnaire de package". The command `PM> Update-Database -Verbose` is entered at the bottom of the console.

```
Console du Gestionnaire de package
Source de packages : Tout | Projet par défaut : MvcOrbusV2
et n'octroie aucune licence les concernant. Certains packages peuvent inclure des licences supplémentaires. Suivez l'URL (flux) de la source de packages pour plus d'informations.

Version de l'hôte de la Console du Gestionnaire de package 4.5.0.4685

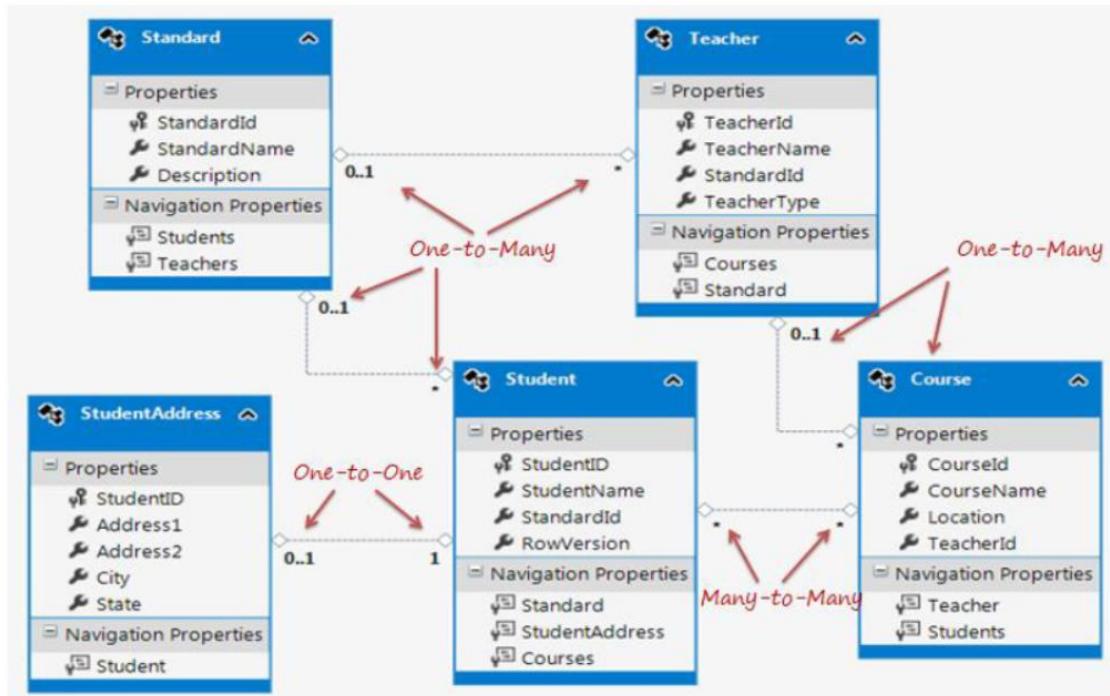
Tapez 'get-help NuGet' pour afficher toutes les commandes NuGet disponibles.

PM> Update-Database -Verbose
181 %
```

Nous allons démontrer le reste avec la création d'une application de gestion d'école élémentaire.

*Exemple*

# Entity Relationships



## Relationship mapping

```

public partial class Student
{
    public Student()
    { this.Courses = new HashSet<Course>(); }

    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public Nullable<int> StandardId { get; set; }
    public byte[] RowVersion { get; set; }

    public virtual Standard Standard { get; set; }
    public virtual StudentAddress StudentAddress { get; set; }
    public virtual ICollection<Course> Courses { get; set; }
}

public partial class Standard
{
    public Standard()
    { this.Students = new HashSet<Student>(); this.Teachers = new HashSet<Teacher>(); }

    public int StandardId { get; set; }
    public string StandardName { get; set; }
    public string Description { get; set; }

    public virtual ICollection<Student> Students { get; set; }
    public virtual ICollection<Teacher> Teachers { get; set; }
}

public partial class Teacher
{
    public Teacher()
    { this.Courses = new HashSet<Course>(); }

    public int TeacherId { get; set; }
    public string TeacherName { get; set; }
    public Nullable<int> StandardId { get; set; }
    public Nullable<int> TeacherType { get; set; }

    public virtual ICollection<Course> Courses { get; set; }
    public virtual Standard Standard { get; set; }
}

public partial class StudentAddress
{
    public int StudentID { get; set; }
    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public string State { get; set; }

    public virtual Student Student { get; set; }
}

public partial class Course
{
    public Course()
    { this.Students = new HashSet<Student>(); }

    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public System.Data.Entity.Spatial.DbGeography Location { get; set; }
    public Nullable<int> TeacherId { get; set; }

    public virtual Teacher Teacher { get; set; }
    public virtual ICollection<Student> Students { get; set; }
}

```

### Le Model

#### Notion sur les DataAnnotations

System.ComponentModel.DataAnnotations Attributes:

Attribute	Description
<b>Key</b>	Marque propriété comme entité clé qui sera mappé à PK de la table liée.
<b>Timestamp</b>	Marquer la propriété comme une colonne d'horodatage non-nullables dans la base de données.
<b>ConcurrencyCheck</b>	ConcurrencyCheck annotation permet de marquer une ou plusieurs propriétés à utiliser pour vérifier la concurrence dans la base de données lorsqu'un utilisateur édite et supprime une entité.
<b>Required</b>	L'annotation requis forcera EF (et MVC) pour veiller à ce que la propriété a des données en elle.
<b>MinLength</b>	MinLength annotation propriété si elle a une longueur minimum de tableau ou une chaîne validée.
<b>MaxLength</b>	MaxLength annotation est la longueur maximale de la propriété qui, à son tour, définit la longueur maximale d'une colonne dans la base de données
<b>StringLength</b>	Indique la longueur minimum et maximum de caractères autorisés dans un champ de données.

System.ComponentModel.DataAnnotations.Schema Attributes:

Attribute	Description
Table	Indiquez le nom de la table DB qui sera mappé avec la classe
Column	Indiquez le nom de la colonne et datatype qui sera mappé avec la propriété
Index	Créer un index pour la colonne spécifiée. (EF 6.1 partir uniquement)
ForeignKey	Spécifiez la propriété clé étrangère pour les biens de navigation
NotMapped	Spécifiez que la propriété ne sera pas mappé avec base de données
DatabaseGenerated	Base de données attribut généré spécifie que la propriété sera mappé à la colonne calculée de la table de base de données. Ainsi, la propriété sera en lecture seule propriété. Elle peut également être

## MVC ASP.NET

utilisée pour mapper la propriété à la colonne d'identité (colonne auto incrémental).

InverseProperty Propriété Inverse est utile lorsque vous avez plusieurs relations entre les deux classes.

ComplexType Mark la classe comme type complexe dans EF.

### Utilisation de validation Annotations

#### Required

```
[Required]  
public string FirstName { get; set; }  
[Required]  
public string LastName { get; set; }
```

#### StringLength

```
[Required]  
[StringLength(160, MinimumLength=3)]  
public string FirstName { get; set; }
```

#### RegularExpression

```
[RegularExpression(@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}")]  
public string Email { get; set; }
```

#### Range

```
[Range(35,44)]  
public int Age { get; set; }  
  
[Range(typeof(decimal), "0.00", "49.99")]  
public decimal Price { get; set; }
```

### Compare

```
[RegularExpression(@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}")]
public string Email { get; set; }
[Compare("Email")]
public string EmailConfirm { get; set; }
```

### Remote

```
[Remote("CheckUserName", "Account")]
public string UserName { get; set; }

public JsonResult CheckUserName(string username)
{
    var result = Membership.FindUsersByName(username).Count == 0;
    return Json(result, JsonRequestBehavior.AllowGet);
}
```

### DataType Enumeration

Member name	Description
CreditCard	Represents a credit card number.
Currency	Represents a currency value.
Custom	Represents a custom data type.
Date	Represents a date value.
DateTime	Represents an instant in time, expressed as a date and time of day.
Duration	Represents a continuous time during which an object exists.
EmailAddress	Represents an e-mail address.
Html	Represents an HTML file.
ImageUrl	Represents a URL to an image.
MultilineText	Represents multi-line text.

Password	Represent a password value.
PhoneNumber	Represents a phone number value.
PostalCode	Represents a postal code.
Text	Represents text that is displayed.
Time	Represents a time value.
Upload	Represents file upload data type.
Url	Represents a URL value.

```
using System;
using System.Web.DynamicData;
using System.ComponentModel.DataAnnotations;

[MetadataType(typeof(CustomerMetaData))]
public partial class Customer
{

}

public class CustomerMetaData
{
    // Add type information.
    [DataType(DataType.EmailAddress)]
    public object EmailAddress;

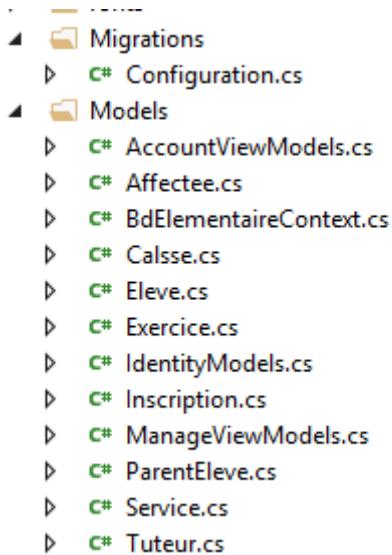
}
```

### Gestion Etablissement élémentaire

Vous avez 3 modules à faire :

- Module Inscription
- Module gestion note
- Module facturation.

### Module Inscription



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
namespace IsiGestionElementaire.Models
{
    public class Eleve
    {
        [Key]
        [ScaffoldColumn(false)]
        public int Id { get; set; }
```

```
[Display(Name = "Matricule"),
 Required(ErrorMessage = "*"), MaxLength(10)]
public string Matricule { get; set; }

[Display(Name = "Nom"),
 Required(ErrorMessage = "*"), MaxLength(40)]
public string Nom { get; set; }

[Display(Name = "Prénom"),
 Required(ErrorMessage = "*"), MaxLength(50)]
public string Prenom { get; set; }

[Display(Name = "sexe"),
 Required(ErrorMessage = "*"), MaxLength(1)]
public string Sexe { get; set; }

[Display(Name = "Date de Naissance"),
 Required(ErrorMessage = "*")]
public DateTime? DateNaiss { get; set; }

[Display(Name = "Lieu de Naissance"),
 Required(ErrorMessage = "*"), MaxLength(80)]
public string LieuNaiss { get; set; }

}

}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
```

```
using System.ComponentModel.DataAnnotations;

namespace IsiGestionElementaire.Models
{
    public class Exercice
    {
        [Key]
        [ScaffoldColumn(false)]
        public int Id { get; set; }

        [Display(Name = "Exercice"), Required(ErrorMessage
= "*"), MaxLength(4)]
        public string Annee { get; set; }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace IsiGestionElementaire.Models
{
    public class ParentEleve
    {
        [Key]
        [Column(Order = 1)]
        public int? IdEleve { get; set; }

        [ForeignKey("IdEleve")]
    }
}
```

```
public virtual Eleve eleve {get; set;}
[Key]
[Column(Order = 2)]
public int? IdTuteur { get; set; }
[ForeignKey("IdTuteur")]
public virtual Tuteur tuteur { get; set; }

[Display(Name = "Priorité"),
 Required(ErrorMessage = "*"), MaxLength(5)]
public string Priorite { get; set; }

}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace IsiGestionElementaire.Models
{
    public class Service
    {
        [Key]
        [ScaffoldColumn(false)]
        public int Id { get; set; }
        [Display(Name = "Libelle"), Required(ErrorMessage = "*"),
        MaxLength(20)]
        public string Libelle { get; set; }
```

---

## MVC ASP.NET

---

```
[Display(Name = "Montant"), Required(ErrorMessage = "*")]
public Double Montant { get; set; }

[Display(Name = "Type"), Required(ErrorMessage = "*"),
MaxLength(20)]
public string Type { get; set; }

}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace IsiGestionElementaire.Models
{
    public class Tuteur
    {
        [Key]
        [ScaffoldColumn(false)]
        public int id { get; set; }

        [Display(Name ="Nom"), MaxLength(80), Required(ErrorMessage
        ="*")]
        public string NomTuteur { get; set; }

        [Display(Name = "Prenom"), MaxLength(80),
        Required(ErrorMessage = "*")]
        public string PrenomTuteur { get; set; }

        [Display(Name = "Adresse"), MaxLength(150),
        Required(ErrorMessage = "*")]
        public string AdresseTuteur { get; set; }
    }
}
```

```
[Display(Name = "Téléphone"), MaxLength(20),
Required(ErrorMessage = "*")]
    public string TelTuteur { get; set; }
[Display(Name = "Email"), MaxLength(80)]
    public string EmailTuteur { get; set; }
[Display(Name = "Civilité"), MaxLength(5),
Required(ErrorMessage = "*")]
    public string CiviliteTuteur { get; set; }
[Display(Name = "Parenté"), MaxLength(20),
Required(ErrorMessage = "*")]
    public string Parente { get; set; }
}
}
```

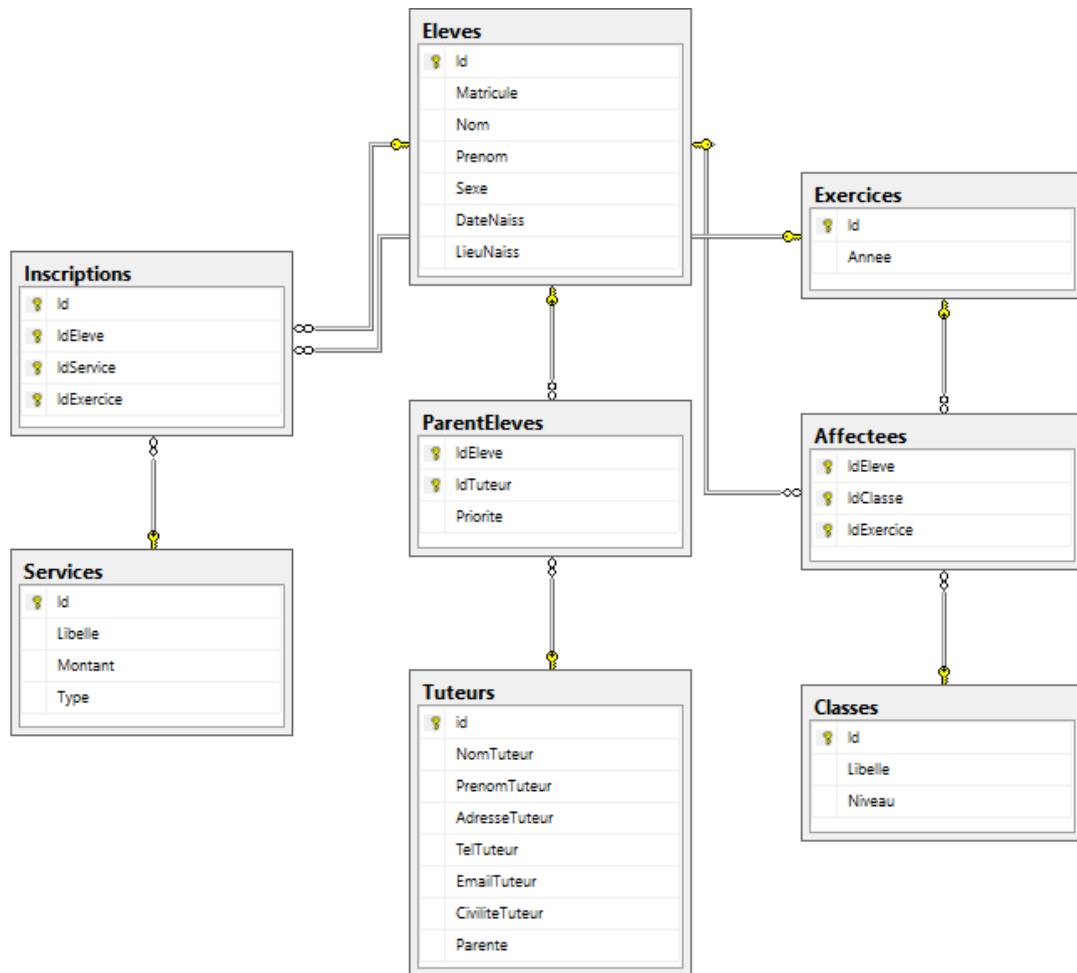
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema

namespace IsiGestionElementaire.Models
{
    public class Affectee
    {
        [Key]
        [Column(Order =1)]
        public int? IdEleve { get; set; }
        [ForeignKey("IdEleve")]
    }
}
```

```
public virtual Eleve eleve {get; set;}  
[Key]  
[Column(Order = 2)]  
public int? IdClasse { get; set; }  
[ForeignKey("IdClasse")]  
public virtual Classe classe { get; set; }  
[Key]  
[Column(Order = 3)]  
public int? IdExercice { get; set; }  
[ForeignKey("IdExercice")]  
public virtual Exercice exercice { get; set; }  
  
}  
}
```

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Web;  
using System.ComponentModel.DataAnnotations;  
using System.ComponentModel.DataAnnotations.Schema;  
  
namespace IsiGestionElementaire.Models  
{  
    public class Inscription  
    {  
        [Key, Column(Order =1)]  
        [ScaffoldColumn(false)]  
        public int Id { get; set; }  
    }  
}
```

```
[Key, Column(Order = 2)]  
public int? IdEleve { get; set; }  
[ForeignKey("IdEleve")]  
public virtual Eleve eleve {get; set;}  
[Key, Column(Order = 3)]  
public int? IdService { get; set; }  
[ForeignKey("IdService")]  
public virtual Service service { get; set; }  
[Key, Column(Order = 4)]  
public int? IdExercice { get; set; }  
[ForeignKey("IdExercice")]  
public virtual Exercice exercice { get; set; }  
  
}  
}
```



Module Facturation

Module Gestion des notes

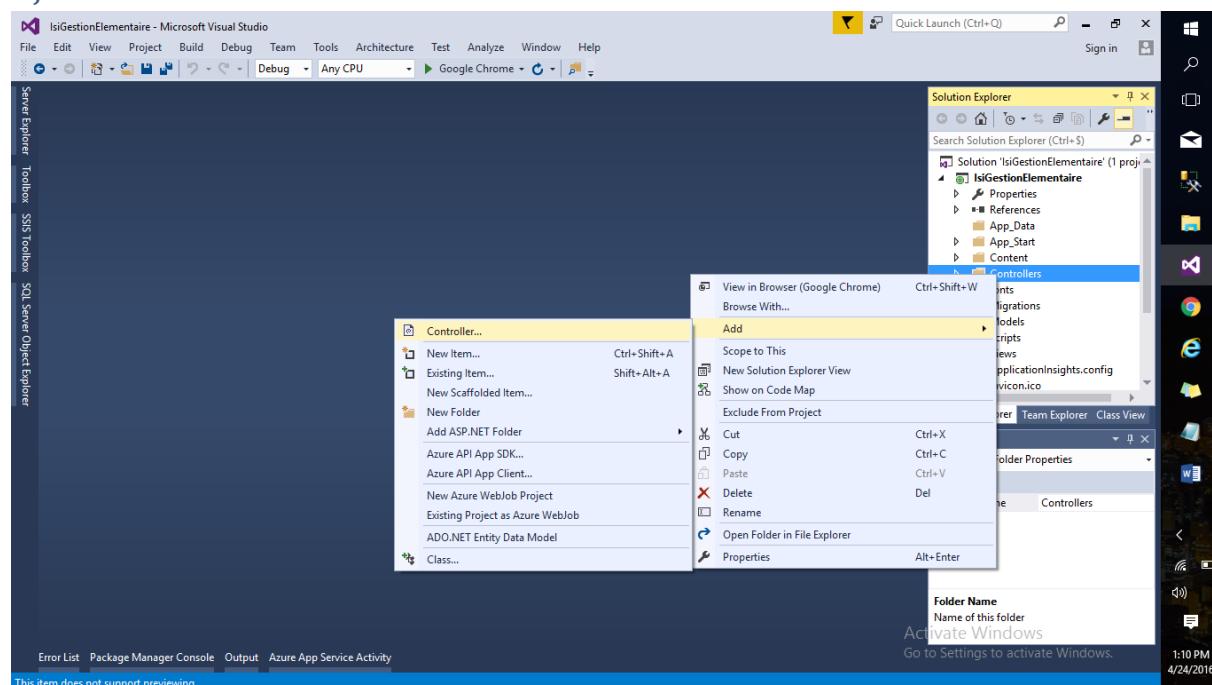
## Les contrôleurs

Le contrôleur de l'architecture MVC gère toute demande d'URL entrante. Le contrôleur est une classe, issue de la *System.Web.Mvc.Controller* de classe de base. Classe Controller contient des méthodes publiques appelées méthodes d'action. Controller et sa méthode d'action pour gérer les requêtes entrantes navigateur, récupère les données des modèles nécessaires et renvoie les réponses appropriées.

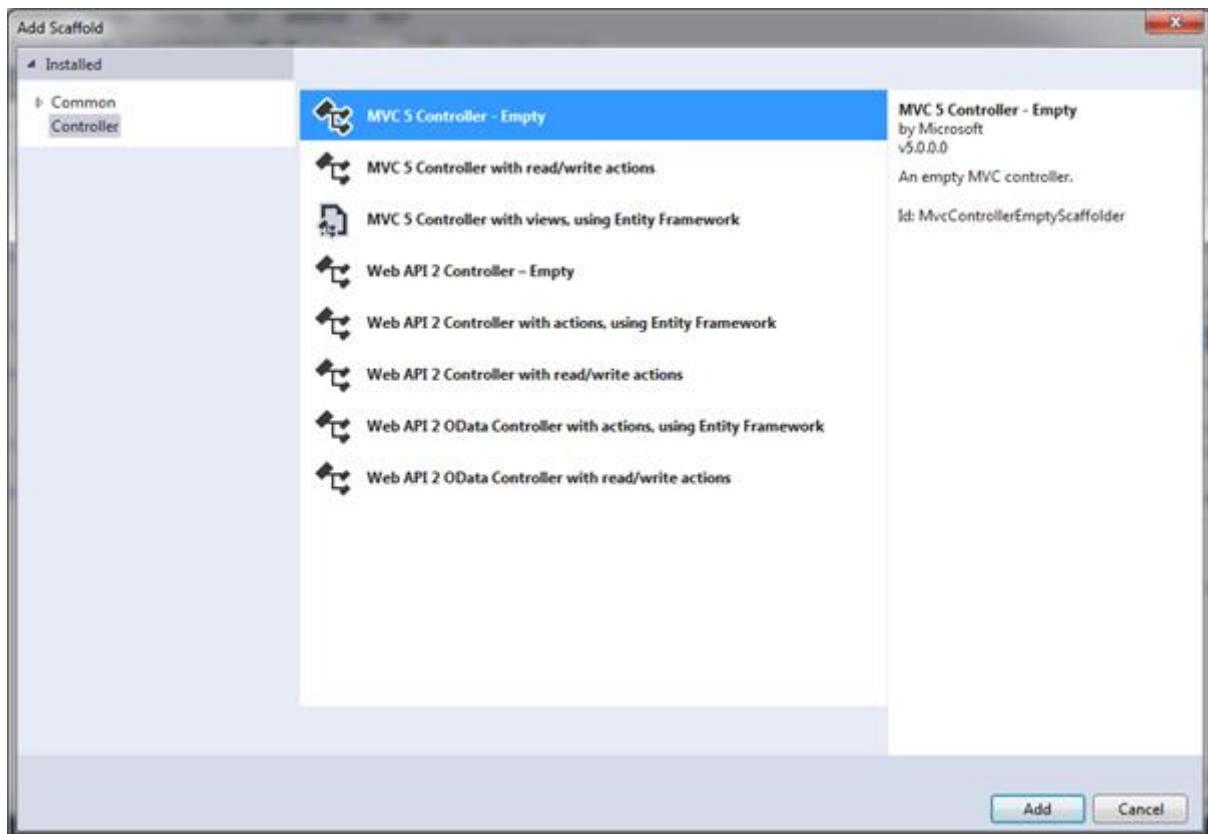
## MVC ASP.NET

Dans ASP.NET MVC, chaque nom de classe de contrôleur doit se terminer par un mot "Controller". Par exemple, contrôleur pour la page d'accueil doit être *HomeController* et contrôleur pour étudiant doit être *StudentController*. En outre, chaque classe de contrôleur doit être située dans le dossier Controller de la structure du dossier MVC.

### Ajout d'un nouveau contrôleur

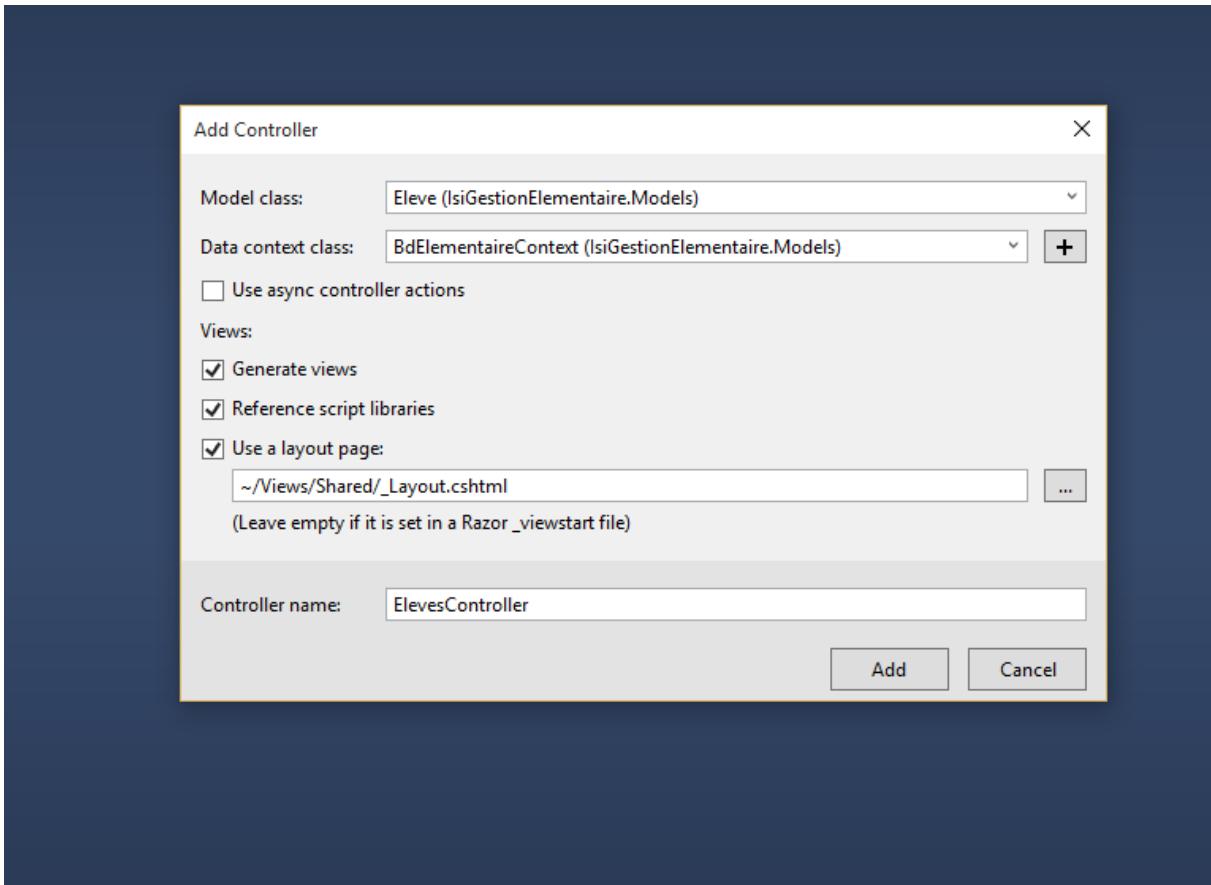


**Remarque :** Scaffolding est un cadre de génération automatique de code pour les applications Web ASP.NET. Scaffolding réduit le temps nécessaire pour développer un contrôleur, voir etc dans framework MVC. Vous pouvez développer un modèle Scaffolding personnalisé en utilisant des modèles T4 selon votre architecture et le codage standard.



Ajouter dialogue Scaffolding contient différents modèles pour créer un nouveau contrôleur. Nous allons en apprendre davantage sur d'autres modèles plus tard.

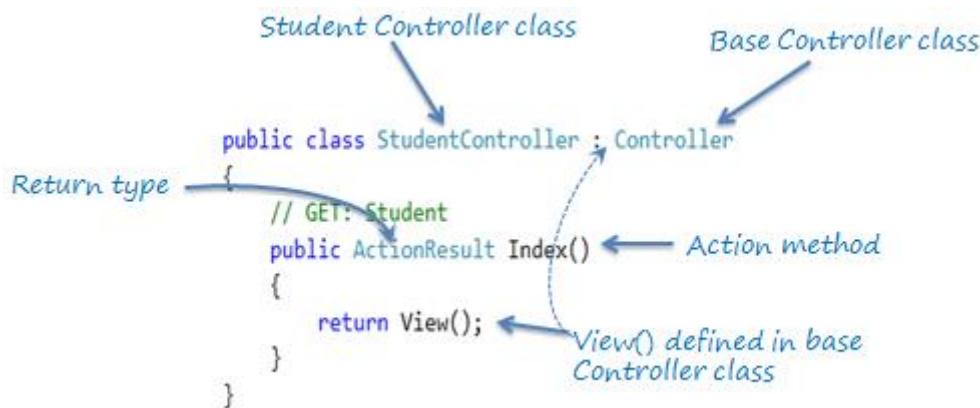
Dans la boîte de dialogue Add Controller, entrez le nom du contrôleur. Rappelez-vous, le nom du contrôleur doit se terminer avec le contrôleur.



## Méthode d'action

Toutes les méthodes publiques d'une classe de contrôleur sont appelées méthodes d'action. Ils sont comme les autres méthodes normales avec les restrictions suivantes :

1. Méthode d'action doit être publique. Il ne peut pas être privé ou protégé
2. Méthode d'action ne peut pas être surchargé
3. Méthode d'action ne peut pas être une méthode statique.



### Méthode d'action par défaut

Chaque contrôleur peut avoir une méthode d'action par défaut comme par itinéraire configuré en classe RouteConfig. Par défaut, Index est une méthode d'action par défaut pour un contrôleur, comme par root par défaut configuré comme illustré ci-dessous.

#### Default Route:

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}/{name}",
    defaults: new { controller = "Home",
                    action = "Index",
                    id = UrlParameter.Optional
    });
}
```

### ActionResult

Framework MVC comprend différentes classes de résultat, qui peut être un retour de méthodes d'action. Ils résultent classes représentent différents types de réponses telles que html, fichier, string, JSON, JavaScript, etc. Le tableau suivant répertorie toutes les classes de résultats disponibles dans ASP.NET MVC.

Result Class	Description
<b>ViewResult</b>	Représente HTML et le balisage.
<b>EmptyResult</b>	Représente Pas de réponse.
<b>ContentResult</b>	Représente chaîne littérale.
<b>FileContentResult/ FilePathResult/ FileStreamResult</b>	Représente le contenu d'un fichier
<b>JavaScriptResult</b>	Représenter un script JavaScript.
<b>JsonResult</b>	Représenter JSON qui peut être utilisé dans Ajax
<b>RedirectResult</b>	Représente une redirection vers une nouvelle URL
<b>RedirectToRouteResult</b>	Représenter une autre action du même ou un autre contrôleur

<b>PartialViewResult</b>	Retourne HTML de la vue partielle
<b>HttpUnauthorizedResult</b>	Retourne HTTP 403 statut

La classe ActionResult est une classe de base de toutes les classes de résultats ci-dessus, il peut donc être le type de retour des méthodes d'action qui renvoie tout type de résultat ci-dessus. Cependant, vous pouvez spécifier la classe de résultat approprié comme un type de méthode d'action de retour.

La méthode View () est définie dans la classe de contrôleur de base. Elle contient également des méthodes différentes, qui retourne automatiquement le type particulier de résultat comme indiqué dans le tableau ci-dessous.

Result Class	Description	Base Controller method
ViewResult	Represents HTML and markup.	View()
EmptyResult	Represents No response.	
ContentResult	Represents string literal.	Content()
FileContentResult, FilePathResult, FileStreamResult	Represents the content of a file	File()
JavaScriptResult	Represent a JavaScript script.	JavaScript()
JsonResult	Represent JSON that can be used in AJAX	Json()
RedirectResult	Represents a redirection to a new URL	Redirect()
RedirectToRouteResult	Represent another action of same or other controller	RedirectToRoute()
PartialViewResult	Returns HTML	PartialView()
HttpUnauthorizedResult	Returns HTTP 403 status	

### Méthode d'action Paramètres

Toutes les méthodes d'action peuvent avoir des paramètres d'entrée comme des méthodes normales. Elle peut être d'un type de données primitives ou des paramètres de type complexes comme indiqué dans l'exemple ci-dessous.

Example: Action method parameters

```
[HttpPost]
public ActionResult Edit(Student std)
{
    // update student to the database

    return RedirectToAction("Index");
}

[HttpDelete]
public ActionResult Delete(int id)
{
    // delete student from the database whose id matches with specified id

    return RedirectToAction("Index");
}
```

Par défaut, les valeurs des paramètres de la méthode d'action sont récupérées à partir de la collecte de données de la demande. La collecte de données comprend le nom / valeurs paires pour les données de formulaire ou des valeurs de chaîne de requête ou valeurs de cookie. Obligatoire dans ASP.NET MVC Modèle mappe automatiquement la collecte des données URL de chaîne de requête ou formulaire pour les paramètres de la méthode d'action si les deux noms sélectionnés. Visitez la section de liaison de modèle pour plus d'informations à ce sujet.

### Sélecteurs action

Le sélecteur d'action est l'attribut qui peut être appliquée aux procédés d'action. Il aide à moteur de routage pour sélectionner la méthode d'action correcte de traiter une demande particulière. MVC 5 comprend les attributs de sélection d'action suivants :

1. ActionName
2. NonAction
3. ActionVerbs

#### ActionName

ActionName attribut nous permet de spécifier un nom d'action différent de celui du nom de la méthode. Prenons l'exemple suivant.

Example: ActionName

```
public class StudentController : Controller
{
    public StudentController()
    {
    }

    [ActionName("find")]
    public ActionResult GetById(int id)
    {
        // get student from the database
        return View();
    }
}
```

#### NonAction

attribut non action de sélection indique qu'une méthode publique d'un contrôleur est pas une méthode d'action. Utilisez l'attribut non action lorsque vous

souhaitez méthode publique dans un contrôleur, mais ne veulent pas de la traiter comme une méthode d'action.

Example: NonAction

```
public class StudentController : Controller
{
    public StudentController()
    {

    }

    [NonAction]
    public Student GetStudent(int id)
    {
        return studentList.Where(s => s.StudentId == id).FirstOrDefault();
    }
}
```

### ActionVerbs

Le sélecteur de ActionVerbs est utilisé lorsque vous souhaitez contrôler la sélection d'une méthode d'action basé sur une méthode de requête Http. Par exemple, vous pouvez définir deux méthodes d'action différentes avec le même nom, mais une méthode d'action répond à une requête HTTP GET et une autre méthode d'action répond à une demande HTTP Post.

framework MVC supporte différents ActionVerbs, comme `HttpGet`, `HttpPost`, `HttpDelete`, `HttpOptions` et `HttpPatch`. Vous pouvez appliquer ces attributs à la méthode d'action pour indiquer le type de demande Http la méthode d'action prend en charge. Si vous ne demandez pas un attribut puis il le juge une requête GET par défaut.

## MVC ASP.NET

http://localhost/Student/Edit/1

HttpGET

```
public ActionResult Edit(int Id)
{
    var std = students.Where(s => s.StudentId == Id).FirstOrDefault();
```

```
        return View(std);
}
```

© TutorialsTeacher.com

http://localhost/Student/Edit

HttpPOST

```
[HttpPost]
public ActionResult Edit(Student std)
{
    //update database here..

    return RedirectToAction("Index");
}
```

Http method      Usage

GET      Pour récupérer les informations à partir du serveur. Les paramètres seront ajoutés dans la chaîne de requête.

POST      Pour créer une nouvelle ressource.

PUT      Pour mettre à jour une ressource existante.

HEAD      Identique à GET, sauf que le serveur ne pas retourner le corps du message.

OPTIONS      méthode OPTIONS représente une demande d'information sur les options de communication pris en charge par le serveur web.

DELETE      Pour supprimer une ressource existante.

PATCH      Pour complète ou mise à jour partielle de la ressource.

### Example: ActionVerbs

```
public class StudentController : Controller
{
```

```
public ActionResult Index()
{
    return View();
}

[HttpPost]
public ActionResult PostAction()
{
    return View("Index");
}

[HttpPut]
public ActionResult PutAction()
{
    return View("Index");
}

[HttpDelete]
public ActionResult DeleteAction()
{
    return View("Index");
}

[HttpHead]
public ActionResult HeadAction()
{
    return View("Index");
}

[HttpOptions]
public ActionResult OptionsAction()
{
    return View("Index");
}

[HttpPatch]
public ActionResult PatchAction()
```

```
{  
    return View("Index");  
}  
}
```

Vous pouvez également appliquer plusieurs verbes HTTP en utilisant AcceptVerbs attribut. Méthode GetAndPostAction prend en charge, GET et POST Verbes d'action dans l'exemple suivant :

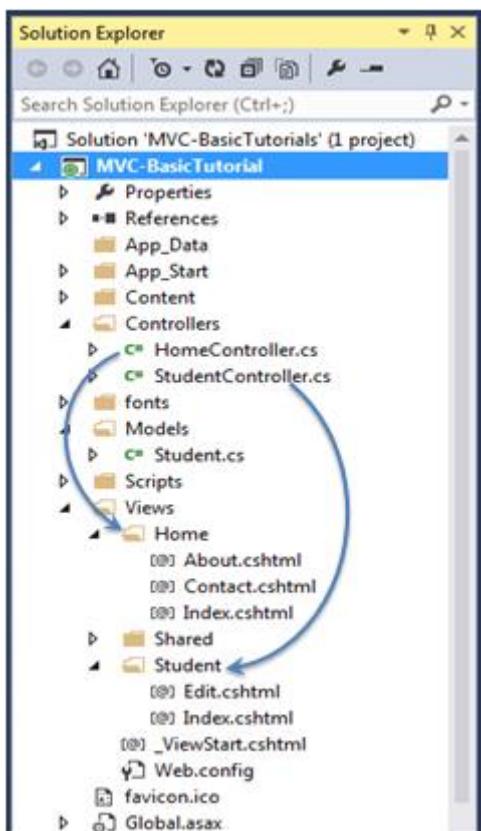
### Example: AcceptVerbs

```
[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Get)]  
public ActionResult GetAndPostAction()  
{  
    return RedirectToAction("Index");  
}
```

## View

La vue est une interface utilisateur. Voir affiche les données du modèle à l'utilisateur et leur permet également de modifier les données.

Vues ASP.NET MVC sont stockés dans le dossier Vues. Différentes méthodes d'une classe unique contrôleur d'action peuvent rendre différents points de vue, de sorte que le dossier Vues contient un dossier séparé pour chaque contrôleur avec le même nom en tant que contrôleur, afin de tenir compte des vues multiples. Par exemple, des vues, qui seront rendus à partir de l'une des méthodes de HomeController d'action, réside dans les vues> Dossier d'accueil. De la même manière.



**Remarque :** le dossier Shared contient des vues, des modèles ou des vues partielles qui seront partagées entre plusieurs points de vue.

### Razor view

Microsoft a introduit le moteur de vue Razor et emballé avec MVC 3. Vous pouvez écrire un mélange de balises HTML et le code côté serveur en vue de rasoir.

Razor utilise le caractère @ pour le serveur de code au lieu de traditionnel <%%> côté. Vous pouvez utiliser C # ou la syntaxe Visual Basic pour écrire le code côté serveur vue à l'intérieur de rasoir. Razor moteur de vue de maximiser la vitesse de l'écriture de code en minimisant le nombre de caractères et de touches requises lors de l'écriture d'une vue. Fichiers vues Razor ont l'extension .cshtml ou vbhtml.

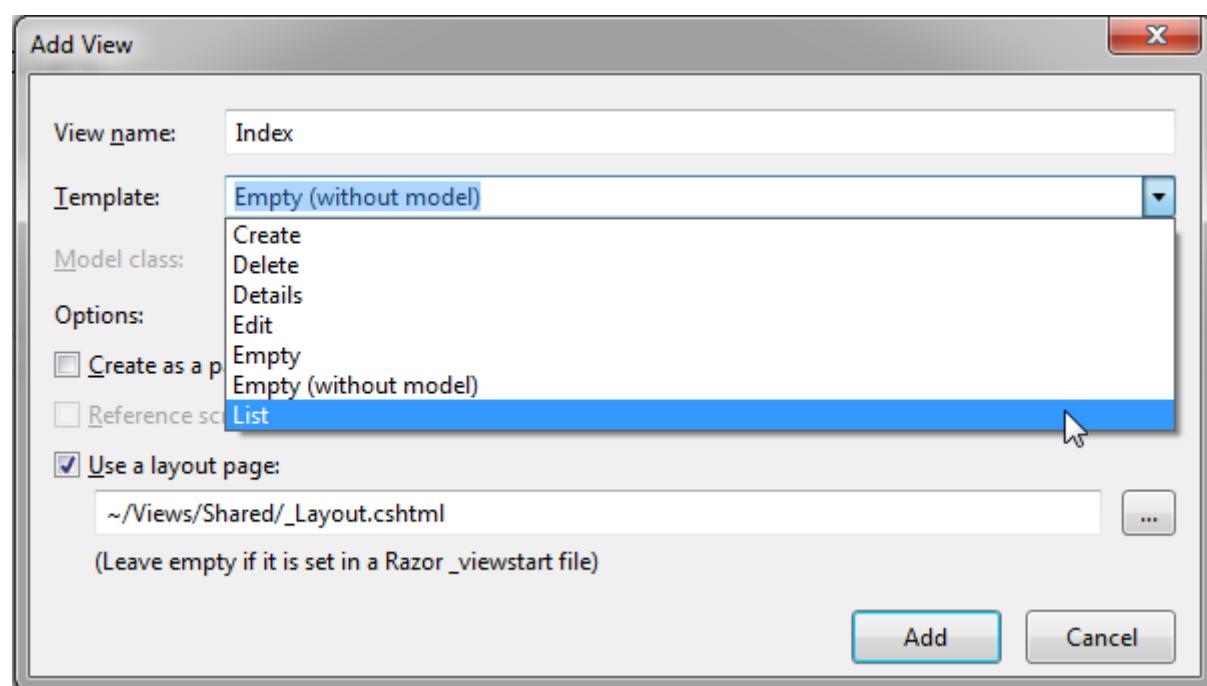
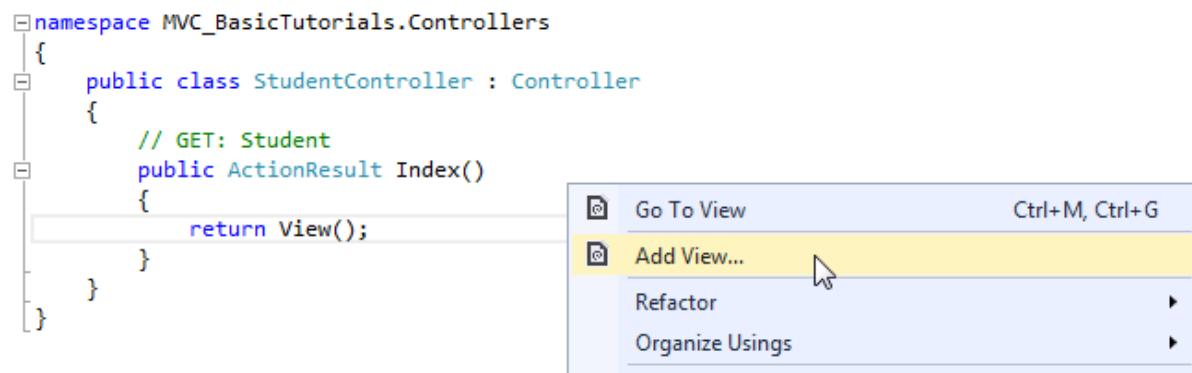
ASP.NET MVC prend en charge les types de fichiers suivants afficher:

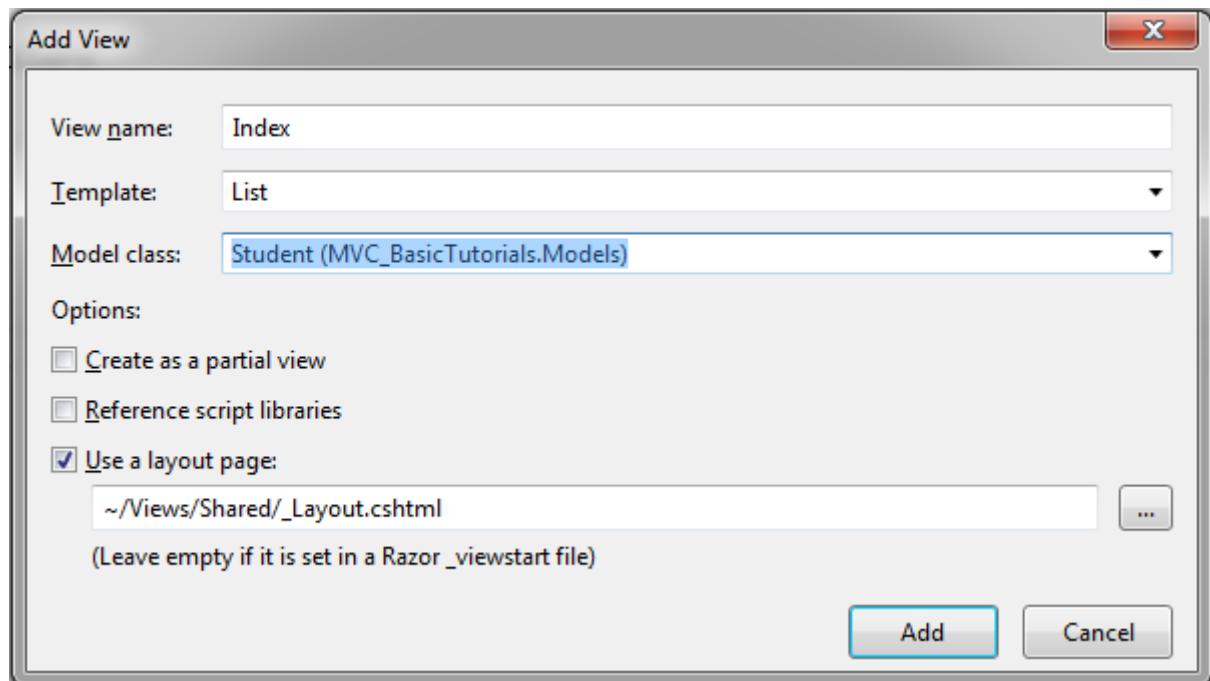
View file extension	Description
.cshtml	Vue C # Razor. Prise en charge de C # avec les balises HTML.

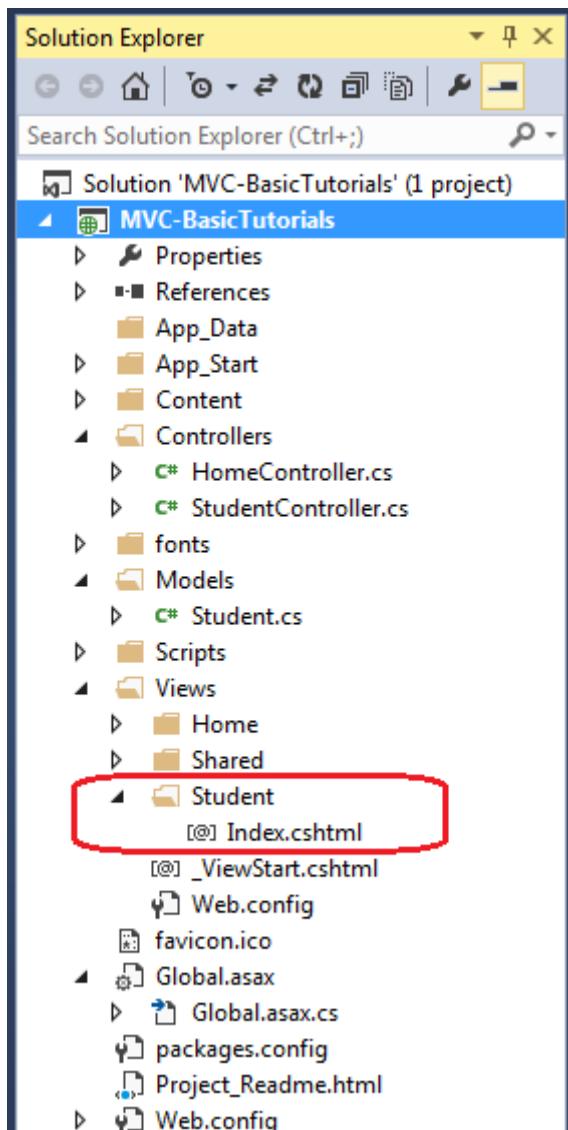
## MVC ASP.NET

.vbhtml	Vue Visual Basic Razor. Prise en charge de Visual Basic avec des balises HTML.
.aspx	ASP.Net web form
.ascx	ASP.NET web control

### Créer une nouvelle View







## Views\Student\Index.cshtml:

```
@model IEnumerable<MVC_BasicTutorials.Models.Student>

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}



## Index



@Html.ActionLink("Create New", "Create")


```

```
</p>



| @Html.DisplayNameFor(model => model.StudentName) | @Html.DisplayNameFor(model => model.Age) |                                                                                                                                                                                                     |
|--------------------------------------------------|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                  |                                          |                                                                                                                                                                                                     |
| @Html.DisplayFor(modelItem => item.StudentName)  | @Html.DisplayFor(modelItem => item.Age)  | @Html.ActionLink("Edit", "Edit", new { id=item.StudentId })   @Html.ActionLink("Details", "Details", new { id=item.StudentId })   @Html.ActionLink("Delete", "Delete", new { id = item.StudentId }) |


```

```
@model IEnumerable<MVC_BasicTutorials.Models.Student>

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}



## Index



@Html.ActionLink\("Create New", "Create"\)



| <a href="#">@Html.DisplayNameFor(model =&gt; model.StudentName)</a> | <a href="#">@Html.DisplayNameFor(model =&gt; model.Age)</a> |                                                                                                                                                                                                                                                   |
|---------------------------------------------------------------------|-------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">@Html.DisplayFor(modelItem =&gt; item.StudentName)</a>  | <a href="#">@Html.DisplayFor(modelItem =&gt; item.Age)</a>  | <a href="#">@Html.ActionLink("Edit", "Edit", new { id=item.StudentId })</a>   <a href="#">@Html.ActionLink("Details", "Details", new { id=item.StudentId })</a>   <a href="#">@Html.ActionLink("Delete", "Delete", new { id=item.StudentId })</a> |


```

*Razor Syntax*

*Html*

*Html helper*

Application name     Home     About     Contact

## Index

Create New

Name	Age	
John	18	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Steve	21	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Bill	25	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ram	20	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Ron	31	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Chris	17	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Rob	19	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

© 2014 - My ASP.NET Application

**Remarque :** Chaque vue dans le ASP.NET MVC est dérivé de la classe WebViewPage inclus dans System.Web.Mvc espace de noms.

### Razor Syntax

Razor est l'un des moteurs de vue pris en charge dans ASP.NET MVC. Razor vous permet d'écrire mélange de HTML et du code côté serveur en utilisant C # ou Visual Basic. vue Razor avec la syntaxe de base visuelle a l'extension de fichier .vbhtml et syntaxe C # a l'extension .cshtml de fichier.

syntaxe Razor a les caractéristiques suivantes:

- **Compact** : syntaxe Razor est compact qui vous permet de minimiser le nombre de caractères et les frappes nécessaires pour écrire un code.
- **Facile à apprendre** : la syntaxe Razor est facile à apprendre où vous pouvez utiliser votre familier langage C # ou Visual Basic.
- **IntelliSense** : syntaxe Razor prend en charge la saisie des instructions dans Visual Studio.

Maintenant, nous allons apprendre comment écrire du code de rasoir.

#### Inline expression

#### C# Razor Syntax:

```
<h1>Razor syntax demo</h1>

<h2>@DateTime.Now.ToShortDateString()</h2>
```

Result:

Razor syntax demo

08-09-2014

#### Multi-statement Code block

#### C# Razor Syntax:

```
@{
    var date = DateTime.Now.ToShortDateString();
```

```
var message = "Hello World";
}

<h2>Today's date is: @date </h2>
<h3>@message</h3>
```

Result:

### Razor syntax demo

Today's date is: 08-09-2014

Hello World!

Display text from code block

Utilisez @: ou <text> / <text> pour afficher les textes dans le bloc de code.

### C# Razor Syntax:

```
@{
    var date = DateTime.Now.ToShortDateString();
    string message = "Hello World!";
    @:Today's date is: @date <br />
    @message
}
```

Result:

### Razor syntax demo

Today's date is: 08-09-2014

Hello World!

### Razor Syntax:

```
@{
```

```
var date = DateTime.Now.ToShortDateString();
string message = "Hello World!";
<text>Today's date is:</text> @date <br />
@message
}
```

Result:

### Razor syntax demo

Today's date is: 08-09-2014

Hello World!

### if-else condition

#### Razor Syntax:

```
@if(DateTime.IsLeapYear(DateTime.Now.Year) )
{
    @DateTime.Now.Year @:is a leap year.
}
else {
    @DateTime.Now.Year @:is not a leap year.
}
```

### for loop

#### Razor Syntax:

```
@for (int i = 0; i < 5; i++) {
    @i.ToString() <br />
}
```

### Model

#### C# Razor Syntax:

```
@model Student
```

```
<h2>Student Detail:</h2>
<ul>
    <li>Student Id: @Model.StudentId</li>
    <li>Student Name: @Model.StudentName</li>
    <li>Age: @Model.Age</li>
</ul>
```

### Declare Variables

#### C# Razor Syntax:

```
@{
    string str = "";

    if(1 > 0)
    {
        str = "Hello World!";
    }
}

<p>@str</p>
```

### HTML Helpers

La classe HtmlHelper génère des éléments HTML à l'aide de l'objet de classe de modèle en vue de razor. Il lie l'objet de modèle à des éléments HTML pour afficher la valeur des propriétés du modèle en éléments html et aussi affecte la valeur des éléments HTML aux propriétés du modèle lors de la soumission du formulaire de contact. Il faut donc toujours utiliser la classe HtmlHelper compte tenu de razor au lieu d'écrire des balises html manuellement.

La figure suivante montre l'utilisation de la classe HtmlHelper dans la vue de razor.



La classe `HtmlHelper` génère des éléments HTML. Par exemple,  
`@Html.ActionList("nouveau", "Create")` générerait ancre tag `<a href="/Student/Create">nouveau</a>`.

<code>HtmlHelper</code>	<code>Strogly Typed HtmlHelpers</code>	Generates Html Control
<code>Html.TextBox</code>	<code>Html.TextBoxFor</code>	Textbox
<code>Html.TextArea</code>	<code>Html.TextAreaFor</code>	TextArea
<code>Html.CheckBox</code>	<code>Html.CheckBoxFor</code>	Checkbox
<code>Html.RadioButton</code>	<code>Html.RadioButtonFor</code>	Radio button
<code>Html.DropDownList</code>	<code>Html.DropDownListFor</code>	Dropdown, combobox
<code>Html.ListBox</code>	<code>Html.ListBoxFor</code>	multi-select list box
<code>Html.Hidden</code>	<code>Html.HiddenFor</code>	Hidden field
<code>Password</code>	<code>Html.PasswordFor</code>	Password textbox
<code>Html.Display</code>	<code>Html.DisplayFor</code>	Html text
<code>Html.Label</code>	<code>Html.LabelFor</code>	Label

## MVC ASP.NET

Html.Editor	Html.EditorFor	Generates Html controls based on data type of specified model property e.g. textbox for string property, numeric field for int, double or other numeric type.
-------------	----------------	---

Create TextBox using HtmlHelper

*TextBox() method signature:*

```
MvcHtmlString Html.TextBox(string name, string value, object htmlAttributes)
```

Example: Html.TextBox() in Razor View

```
@model Student  
  
@Html.TextBox("StudentName", null, new { @class = "form-control" })
```

Html Result:

```
<input class="form-control"  
      id="StudentName"  
      name="StudentName"  
      type="text"  
      value="" />
```

Example: Html.TextBox() in Razor View

```
@Html.TextBox("myTextBox", "This is value", new { @class = "form-control" })
```

Html Result:

```
<input class="form-control"  
      id="myTextBox"  
      name="myTextBox"  
      type="text"
```

```
value="This is value" />
```

### TextBoxFor() method Signature:

```
MvcHtmlString TextBoxFor(Expression<Func<TModel, TValue>> expression, object htmlAttributes)
```

### Example: TextBoxFor() in Razor View

```
@model Student
```

```
@Html.TextBoxFor(m => m.StudentName, new { @class = "form-control" })
```

### Html Result:

```
<input class="form-control"
       id="StudentName"
       name="StudentName"
       type="text"
       value="John" />
```

### Différence entre TextBox et TextBoxFor:

- @Html.TextBox() est vaguement méthode alors que @Html.TextBoxFor() est un (générique) méthode d'extension fortement typé.
- TextBox() nécessite nom de la propriété comme paramètre de chaîne alors que TextBoxFor() nécessite l'expression lambda en tant que paramètre.
- TextBox ne donne pas vous compilez erreur de temps si vous avez spécifié un mauvais nom de la propriété.
- TextBoxFor est méthode générique de sorte qu'il donnera vous compilez erreur de temps si vous avez spécifié le nom de la propriété ou le nom de la propriété des modifications erronées.

Create TextArea using HtmlHelper

*TextBox() method Signature:*

```
MvcHtmlString Html.TextArea(string name, string value, object htmlAttributes)
```

Example: Html.TextArea() in Razor View

```
@model Student
```

```
@Html.TextArea("Description", null, new { @class = "form-control" })
```

Html Result:

```
<textarea class="form-control"
          id="Description"
          name="Description"
          rows="2"
          cols="20">This is value</textarea>
```

Example: Html.TextArea() in Razor View

```
@Html.TextArea("myTextArea", "This is value", new { @class = "form-control" })
```

Html Result:

```
<textarea class="form-control"
          cols="20"
          id="myTextArea"
          name="myTextArea"
          rows="2">This is value</textarea>
```

*TextBoxFor() method Signature:*

```
MvcHtmlString TextAreaFor(<Expression<Func<TModel, TValue>> expression,
                           object htmlAttributes)
```

### Example: TextAreaFor() in Razor View

```
@model Student
```

```
@Html.TextAreaFor(m => m.Description, new { @class = "form-control" })
```

Html Result:

```
<textarea class="form-control"  
         cols="20"  
         id="Description"  
         name="Description"  
         rows="2"></textarea>
```

### Create CheckBox using HtmlHelper

*CheckBox() method Signature:*

*MvcHtmlString CheckBox(string name, bool isChecked, object htmlAttributes)*

### Example: Html.CheckBox() in Razor View

```
@Html.CheckBox("isNewlyEnrolled", true)
```

Html Result:

```
<input checked="checked"  
       id="isNewlyEnrolled"  
       name="isNewlyEnrolled"  
       type="checkbox"  
       value="true" />
```

*CheckBoxFor() method Signature:*

*MvcHtmlString CheckBoxFor(<Expression<Func<TModel, TValue>> expression,  
object htmlAttributes)*

### Example: Html.CheckBoxFor() in Razor View

```
@model Student
```

```
@Html.CheckBoxFor(m => m.isNewlyEnrolled)
```

Html Result:

```
<input data-val="true"
       data-val-required="The isNewlyEnrolled field is required."
       id="isNewlyEnrolled"
       name="isNewlyEnrolled"
       type="checkbox"
       value="true" />

<input name="isNewlyEnrolled" type="hidden" value="false" />
```

Create RadioButton using HtmlHelper

*RadioButton() method Signature:*

```
MvcHtmlString RadioButton(string name, object value, bool isChecked, object
htmlAttributes)
```

Example: Html.RadioButton() in Razor View

```
Male: @Html.RadioButton("Gender", "Male")
Female: @Html.RadioButton("Gender", "Female")
```

Html Result:

```
Male: <input checked="checked"
           id="Gender"
           name="Gender"
           type="radio"
           value="Male" />
```

```
Female: <input id="Gender"
```

```
name="Gender"  
type="radio"  
value="Female" />
```

**RadioButtonFor() method Signature:**

```
MvcHtmlString RadioButtonFor(<Expression<Func<TModel, TValue>> expression,  
object value, object htmlAttributes)
```

Example: Html.RadioButtonFor() in Razor View

```
@model Student  
  
@Html.RadioButtonFor(m => m.Gender, "Male")  
@Html.RadioButtonFor(m => m.Gender, "Female")
```

Html Result:

```
<input checked="checked"  
      id="Gender"  
      name="Gender"  
      type="radio"  
      value="Male" />  
  
<input id="Gender"  
      name="Gender"  
      type="radio"  
      value="Female" />
```

Create DropDownList using HtmlHelper

**DropDownList() method signature:**

```
MvcHtmlString Html.DropDownList(string name, IEnumerable<SelectListItem>  
selectList, string optionLabel, object htmlAttributes)
```

Example: Html.DropDownList() in Razor View

```
@model Student

@Html.DropDownList("StudentGender",
    new SelectList(Enum.GetValues(typeof(Gender))),
    "Select Gender",
    new { @class = "form-control" })
```

Html Result:

```
<select class="form-control" id="StudentGender" name="StudentGender">

    <option>Select Gender</option>

    <option>Male</option>

    <option>Female</option>

</select>
```

*DropDownListFor() method signature:*

```
MvcHtmlString Html.DropDownListFor(Expression<Func<dynamic,TProperty>>
expression, IEnumerable<SelectListItem> selectList, string optionLabel, object
htmlAttributes)
```

Example: DropDownListFor() in Razor View

```
@model Student

@Html.DropDownListFor(m => m.StudentGender,
    new SelectList(Enum.GetValues(typeof(Gender))),
    "Select Gender")
```

Html Result:

```
<select class="form-control" id="StudentGender" name="StudentGender">

    <option>Select Gender</option>

    <option>Male</option>

    <option>Female</option>
```

```
</select>
```

Create Hidden field using HtmlHelper

**Hidden() method signature:**

```
MvcHtmlString Html.Hidden(string name, object value, object htmlAttributes)
```

Example: Html.Hidden() in Razor View

```
@model Student  
  
@Html.Hidden("StudentId")  
Html Result:  
<input id="StudentId"  
       name="StudentId"  
       type="hidden"  
       value="1" />
```

**HiddenFor() method signature:**

```
MvcHtmlString          Html.HiddenFor(Expression<Func<dynamic, TProperty>>  
expression)
```

Example: HiddenFor() in Razor View

```
@model Student  
  
@Html.HiddenFor(m => m.StudentId)  
Html Result:  
<input data-val="true"  
       data-val-number="The field StudentId must be a number."  
       data-val-required="The StudentId field is required."  
       id="StudentId"  
       name="StudentId"  
       type="hidden"  
       value="1" />
```

### Create Password field using HtmlHelper

#### Password() method signature:

MvcHtmlString Html.Password(string name, object value, object htmlAttributes)

### Example: Html.Password() in Razor View

```
@model Student

@Html.Password("OnlinePassword")
```

Html Result:

```
<input id="OnlinePassword"
       name="OnlinePassword"
       type="password"
       value="" />
```

#### PasswordFor() method signature:

MvcHtmlString Html.PasswordFor(Expression<Func<dynamic, TProperty>> expression, object htmlAttributes)

### Example: PasswordFor() in Razor View

```
@model Student

@Html.PasswordFor(m => m.Password)
```

Html Result:

```
<input id="Password" name="Password" type="password" value="mypassword" />
```

### Create Html String using HtmlHelper

Display() method Signature: MvcHtmlString Display(string expression)

### Example: Html.Display() in Razor View

```
@Html.Display("StudentName")
```

Html Result:

```
"Steve"
```

DisplayFor() method Signature: `MvcHtmlString`

```
DisplayFor(<Expression<Func<TModel, TValue>> expression)
```

Example: PasswordFor() in Razor View

```
@model Student
```

```
@Html.DisplayFor(m => m.StudentName)
```

Html Result:

```
" Steve "
```

Create Label using HtmlHelper

Label() method Signature: `MvcHtmlString Label(string expression, string labelText, object htmlAttributes)`

Example: Html.Label() in Razor View

```
@Html.Label("StudentName")
```

Html Result:

```
<label for="StudentName">Name</label>
```

Example: Html.Label() in Razor View

```
@Html.Label("StudentName", "Student-Name")
```

Html Result:

```
<label for="StudentName">Student-Name</label>
```

LabelFor() method Signature: `MvcHtmlString`

```
LabelFor(<Expression<Func<TModel, TValue>> expression)
```

Example: LabelFor() in Razor View

```
@model Student
```

```
@Html.LabelFor(m => m.StudentName)
```

Html Result:

```
<label for="StudentName">Name</label>
```

### HtmlHelper.Editor

Nous avons vu différentes méthodes de HtmlHelper utilisées pour les éléments HTML différents générés dans les sections précédentes. ASP.NET MVC comprend également un procédé qui génère des éléments d'entrée html sur la base du type de données. Editor () ou EditorFor () méthode d'extension génère des éléments html en fonction du type de la propriété de l'objet de modèle de données.

La liste de tableau suivant l'élément html créé pour chaque type de données par l'éditeur () ou EditorFor () méthode.

Property DataType	Html Element
string	<input type="text" >
int	<input type="number" >
decimal, float	<input type="text" >
boolean	<input type="checkbox" >
Enum	<input type="text" >
DateTime	<input type="datetime" >

Editor() signature: `MvcHtmlString Editor(string propertname)`

### Example: Editor() in Razor view

```
StudentId:      @Html.Editor("StudentId")
Student Name:   @Html.Editor("StudentName")
Age:           @Html.Editor("Age")
Password:       @Html.Editor("Password")
isNewlyEnrolle: @Html.Editor("isNewlyEnrolle")
Gender:         @Html.Editor("Gender")
DoB:            @Html.Editor("DoB")
```

StudentId:	<input type="text" value="1"/>
Student Name:	<input type="text" value="Jogn"/>
Age:	<input type="text" value="19"/>
Password:	<input type="text" value="sdf"/>
isNewlyEnrolled:	<input checked="" type="checkbox"/>
Gender:	<input type="text" value="Boy"/>
DoB:	<input type="text" value="02-06-2015 11:39:15"/>

EditorFor() signature: `MvcHtmlString EditorFor(<Expression<Func<TModel, TValue>> expression)`

### Example: EditorFor() in Razor view

```
StudentId:      @Html.EditorFor(m => m.StudentId)
Student Name:   @Html.EditorFor(m => m.StudentName)
Age:           @Html.EditorFor(m => m.Age)
Password:       @Html.EditorFor(m => m.Password)
isNewlyEnrolled: @Html.EditorFor(m => m.isNewlyEnrolled)
Gender:         @Html.EditorFor(m => m.Gender)
DoB:            @Html.EditorFor(m => m.DoB)
```

StudentId:	<input type="text" value="1"/>
Student Name:	<input type="text" value="Jogn"/>
Age:	<input type="text" value="19"/>
Password:	<input type="text" value="sdf"/>
isNewlyEnrolled:	<input checked="" type="checkbox"/>
Gender:	<input type="text" value="Boy"/>
DoB:	<input type="text" value="02-06-2015 11:39:15"/>

## Pratique controller/ View avec gestion etablissement Elementaire.

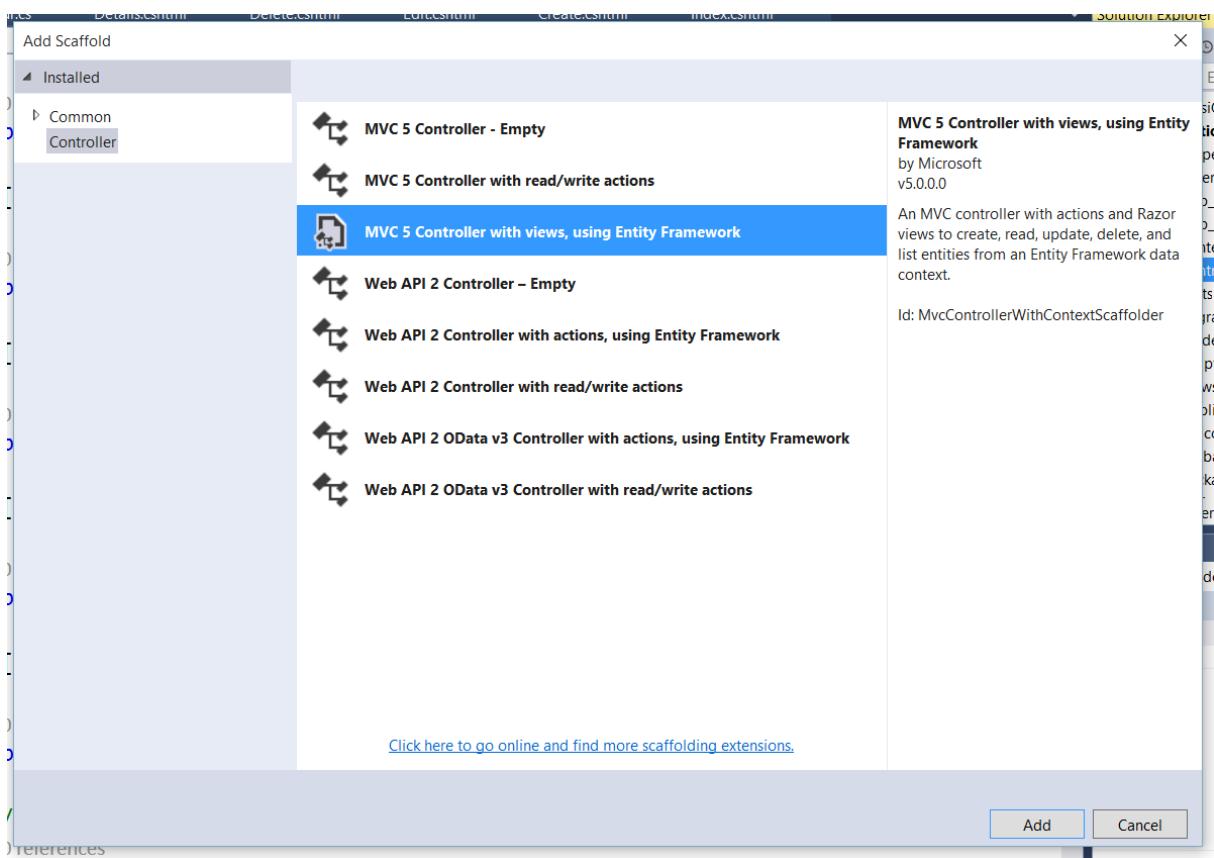
Nous allons d'abord créer des Controller avec les classes Tuteurs, Classes, Exercices et services. Ces derniers sont des formulaires indépendants.

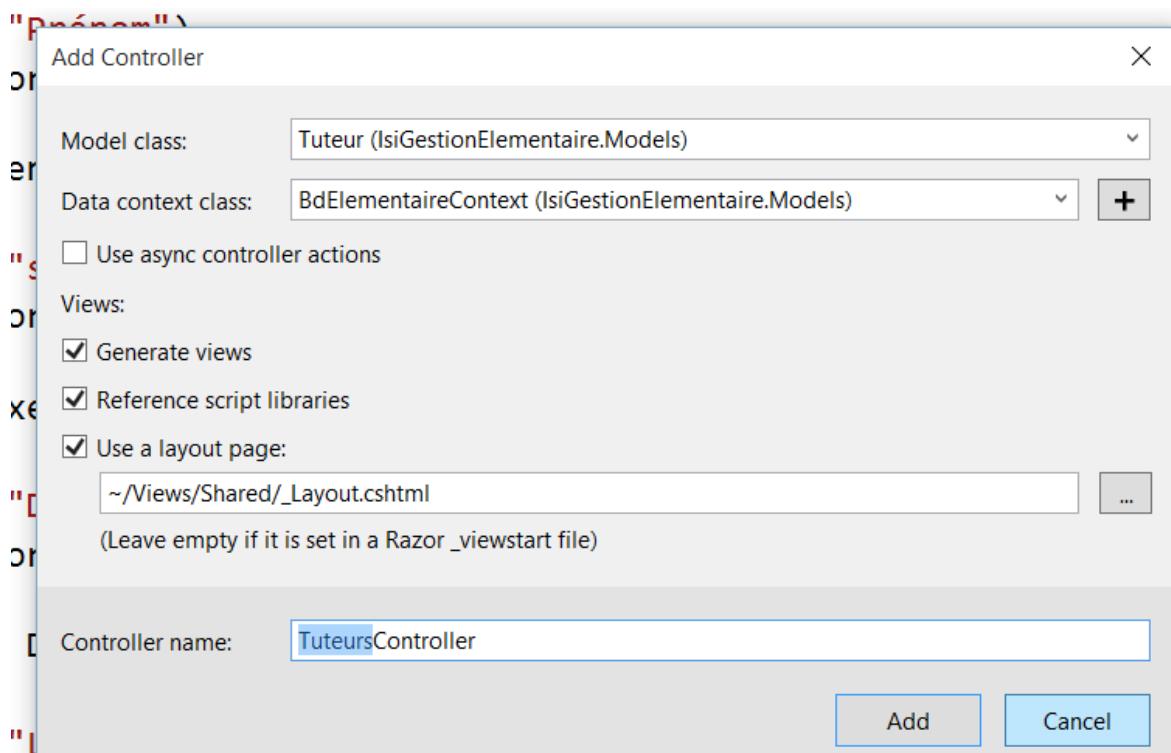
# MVC ASP.NET

```

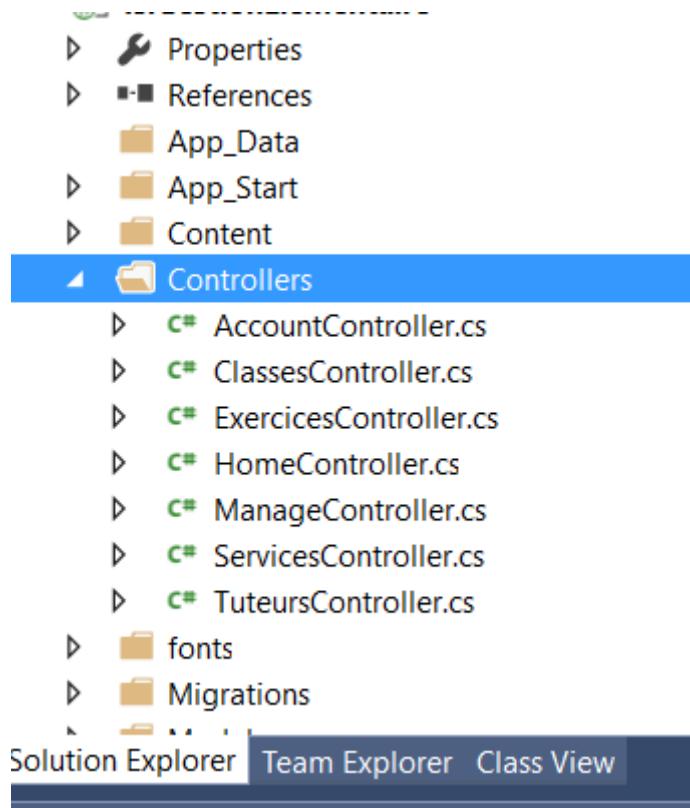
50     [Required(ErrorMessage = "#"), MaxLength(40)]
51     public string Nom { get; set; }
52
53     [Display(Name = "Prénom"),
54      Required(ErrorMessage = "*"), MaxLength(50)]
55     public string Prenom { get; set; }
56
57     [Display(Name = "sexe"),
58      Required(ErrorMessage =
59     public string Sexe { get;
60
61     [Display(Name = "Date de N."), Required(ErrorMessage =
62     public DateTime? DateNaiss { get;
63
64     [Display(Name = "Lieu de Naiss."), Required(ErrorMessage =
65     public string LieuNaiss { get; set; }
66
67     //Espace Tuteur
68
69     public int idTuteur { get; set; }

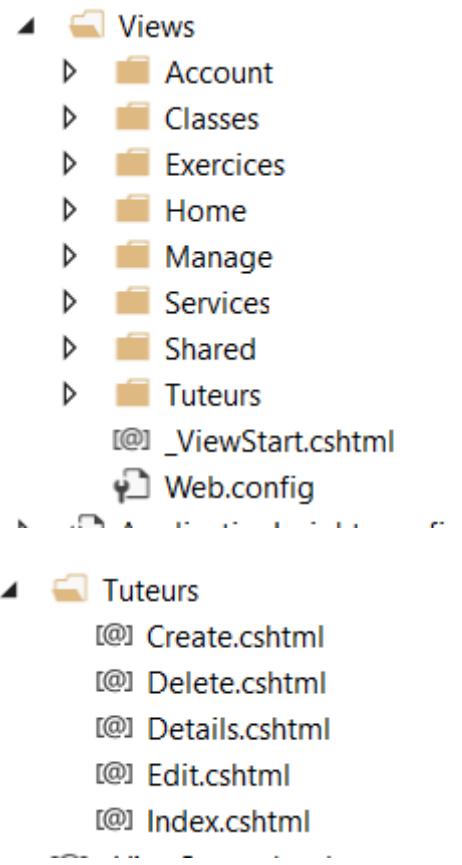
```





```
orMessage = "*" ). MaxLength(80) ]
```





```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Web;
using System.Web.Mvc;
using IsiGestionElementaire.Models;

namespace IsiGestionElementaire.Controllers
{
    public class TuteursController : Controller
    {
        private BdElementaireContext db = new BdElementaireContext();

        // GET: Tuteurs
        public ActionResult Index()
        {
            return View(db.Tuteur.ToList());
        }

        // GET: Tuteurs/Details/5
        public ActionResult Details(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
```

```
        }
        Tuteur tuteur = db.Tuteur.Find(id);
        if (tuteur == null)
        {
            return HttpNotFound();
        }
        return View(tuteur);
    }

    // GET: Tuteurs/Create
    public ActionResult Create()
    {
        return View();
    }

    // POST: Tuteurs/Create
    // To protect from overposting attacks, please enable the specific properties
    you want to bind to, for
    // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create([Bind(Include =
    "id,NomTuteur,PrenomTuteur,AdresseTuteur,TelTuteur,EmailTuteur,CivilitéTuteur,Parente")]
    Tuteur tuteur)
    {
        if (ModelState.IsValid)
        {
            db.Tuteur.Add(tuteur);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        return View(tuteur);
    }

    // GET: Tuteurs/Edit/5
    public ActionResult Edit(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Tuteur tuteur = db.Tuteur.Find(id);
        if (tuteur == null)
        {
            return HttpNotFound();
        }
        return View(tuteur);
    }

    // POST: Tuteurs/Edit/5
    // To protect from overposting attacks, please enable the specific properties
    you want to bind to, for
    // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Edit([Bind(Include =
    "id,NomTuteur,PrenomTuteur,AdresseTuteur,TelTuteur,EmailTuteur,CivilitéTuteur,Parente")]
    Tuteur tuteur)
    {
```

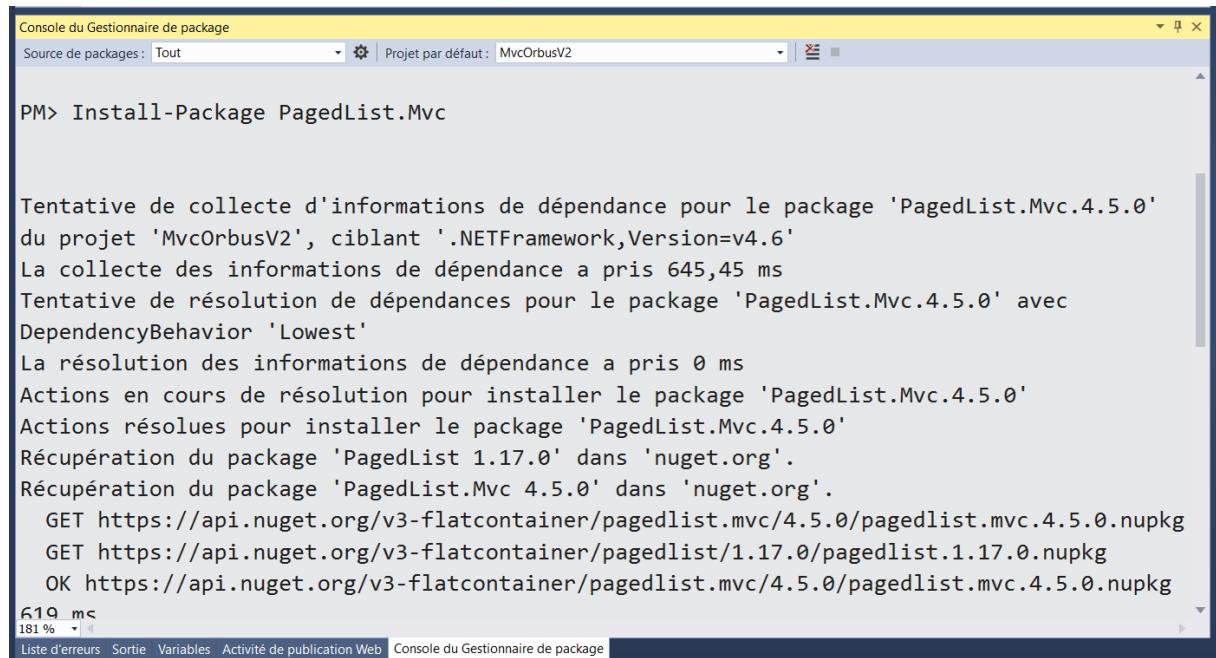
```
    if (ModelState.IsValid)
    {
        db.Entry(tuteur).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(tuteur);
}

// GET: Tuteurs/Delete/5
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Tuteur tuteur = db.Tuteur.Find(id);
    if (tuteur == null)
    {
        return HttpNotFound();
    }
    return View(tuteur);
}

// POST: Tuteurs/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    Tuteur tuteur = db.Tuteur.Find(id);
    db.Tuteur.Remove(tuteur);
    db.SaveChanges();
    return RedirectToAction("Index");
}

protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        db.Dispose();
    }
    base.Dispose(disposing);
}
}
```

## Installation et mise en œuvre des pagedList



The screenshot shows the NuGet Package Manager Console window. The command entered is "PM> Install-Package PagedList.Mvc". The output log displays the process of collecting dependency information for the package 'PagedList.Mvc.4.5.0' in the project 'MvcOrbusV2'. It shows actions like GET requests for the package files from nuget.org and the successful download of the package.

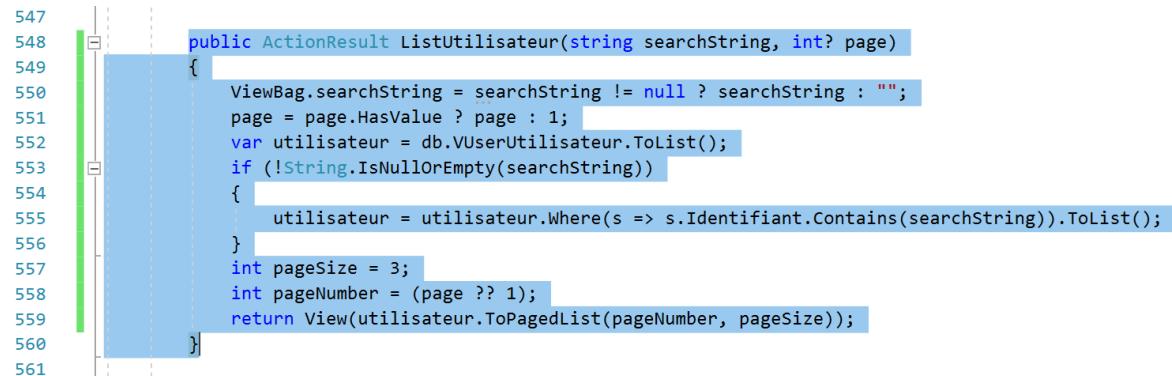
```
PM> Install-Package PagedList.Mvc

Tentative de collecte d'informations de dépendance pour le package 'PagedList.Mvc.4.5.0'
du projet 'MvcOrbusV2', ciblant '.NETFramework,Version=v4.6'
La collecte des informations de dépendance a pris 645,45 ms
Tentative de résolution de dépendances pour le package 'PagedList.Mvc.4.5.0' avec
DependencyBehavior 'Lowest'
La résolution des informations de dépendance a pris 0 ms
Actions en cours de résolution pour installer le package 'PagedList.Mvc.4.5.0'
Actions résolues pour installer le package 'PagedList.Mvc.4.5.0'
Récupération du package 'PagedList 1.17.0' dans 'nuget.org'.
Récupération du package 'PagedList.Mvc 4.5.0' dans 'nuget.org'.
GET https://api.nuget.org/v3-flatcontainer/pagedlist.mvc/4.5.0/pagedlist.mvc.4.5.0.nupkg
GET https://api.nuget.org/v3-flatcontainer/pagedlist/1.17.0/pagedlist.1.17.0.nupkg
OK https://api.nuget.org/v3-flatcontainer/pagedlist.mvc/4.5.0/pagedlist.mvc.4.5.0.nupkg
```

Référencer le Css de pagedList dans le bundle.

```
bundles.Add(new StyleBundle("~/Content/css").Include(
    "~/Content/bootstrap.css",
    "~/Content/PagedList.css",
    "~/Content/site.css"));
```

Au niveau du Controller, vous aurez besoin du nombre d'élément par page, de gérer la page courante

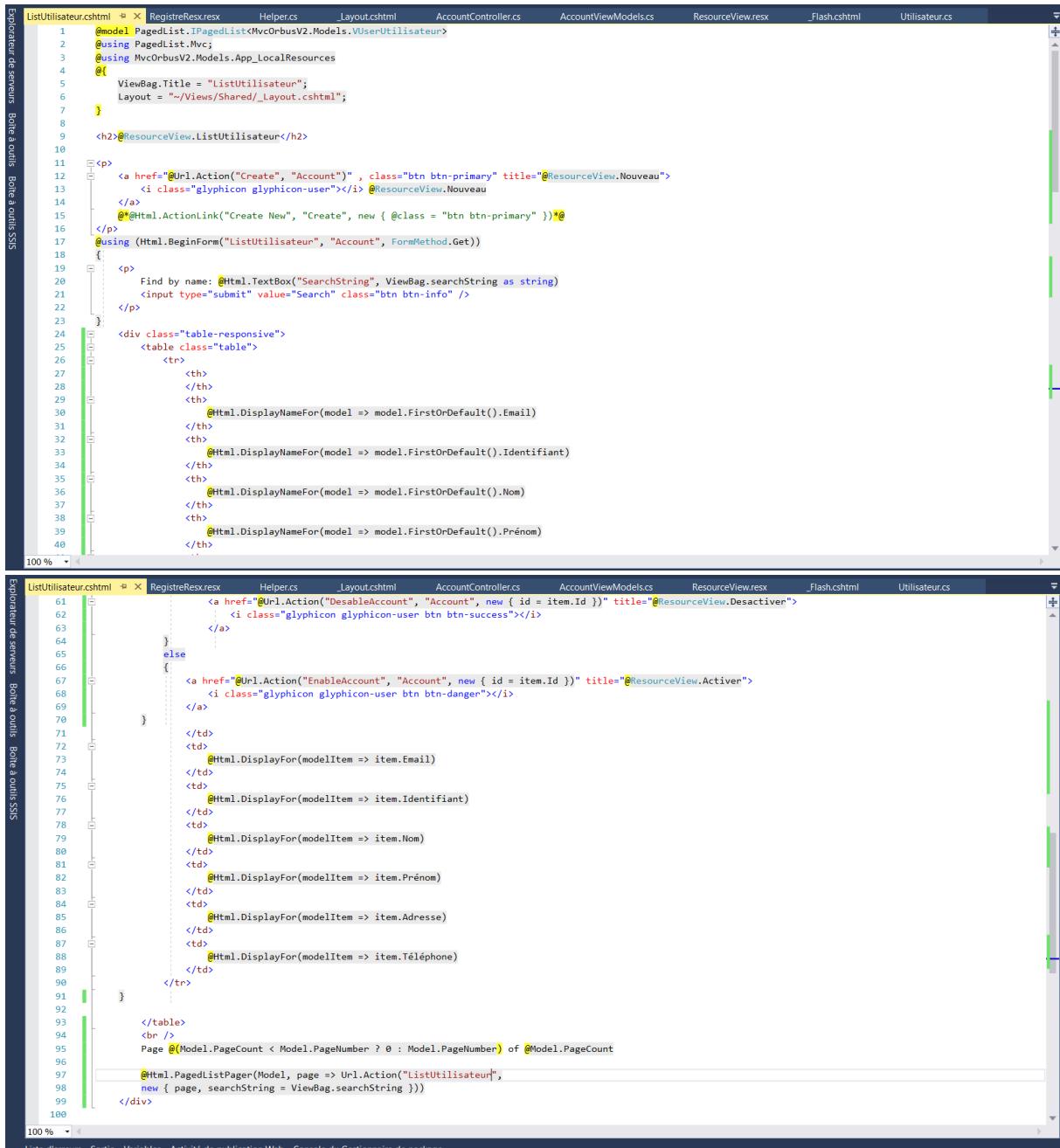


The screenshot shows a portion of a C# code editor. A specific line of code is highlighted: "return View(utilisateur.ToPagedList(pageNumber, pageSize));". This line is part of a controller action named "ListUtilisateur". The code handles a search string and a page number to filter and paginate the user list.

```
547
548     public ActionResult ListUtilisateur(string searchString, int? page)
549     {
550         ViewBag.searchString = searchString != null ? searchString : "";
551         page = page.HasValue ? page : 1;
552         var utilisateur = db.VUserUtilisateur.ToList();
553         if (!String.IsNullOrEmpty(searchString))
554         {
555             utilisateur = utilisateur.Where(s => s.Identifiant.Contains(searchString)).ToList();
556         }
557         int pageSize = 3;
558         int pageNumber = (page ?? 1);
559         return View(utilisateur.ToPagedList(pageNumber, pageSize));
560     }
561 }
```

Référencer pagedList sur le model

# MVC ASP.NET



```
1 @model PagedList.IPagedList<MvcOrbusV2.Models.Utilisateur>
2 @using PagedList.Mvc;
3 @using MvcOrbusV2.Models.App_LocalResources
4 @{
5     ViewBag.Title = "ListUtilisateur";
6     Layout = "~/Views/Shared/_Layout.cshtml";
7 }
8
9 <h2>@ResourceView.ListUtilisateur</h2>
10
11 <p>
12     <a href="@Url.Action("Create", "Account")" class="btn btn-primary" title="@ResourceView.Nouveau">
13         <i class="glyphicon glyphicon-user"></i> @ResourceView.Nouveau
14     </a>
15     @Html.ActionLink("Create New", "Create", new { @class = "btn btn-primary" })
16 </p>
17 using (Html.BeginForm("ListUtilisateur", "Account", FormMethod.Get))
18 {
19
20     Find by name: @Html.TextBox("SearchString", ViewBag.searchString as string)
21     <input type="submit" value="Search" class="btn btn-info" />
22 </p>
23
24 <div class="table-responsive">
25     <table class="table">
26         <tr>
27             <th>
28             <th>
29                 @Html.DisplayNameFor(model => model.FirstOrDefault().Email)
30             </th>
31             <th>
32                 @Html.DisplayNameFor(model => model.FirstOrDefault().Identifiant)
33             </th>
34             <th>
35                 @Html.DisplayNameFor(model => model.FirstOrDefault().Nom)
36             </th>
37             <th>
38                 @Html.DisplayNameFor(model => model.FirstOrDefault().Prénom)
39             </th>
40             ...
41         </tr>
42         <tr>
43             <td>
44                 <a href="@Url.Action("DesableAccount", "Account", new { id = item.Id })" title="@ResourceView.Desactiver">
45                     <i class="glyphicon glyphicon-user btn btn-success"></i>
46                 </a>
47             </td>
48             <td>
49                 @Html.DisplayFor(modelItem => item.Email)
50             </td>
51             <td>
52                 @Html.DisplayFor(modelItem => item.Identifiant)
53             </td>
54             <td>
55                 @Html.DisplayFor(modelItem => item.Nom)
56             </td>
57             <td>
58                 @Html.DisplayFor(modelItem => item.Prénom)
59             </td>
60             <td>
61                 @Html.DisplayFor(modelItem => item.Adresse)
62             </td>
63             <td>
64                 @Html.DisplayFor(modelItem => item.Téléphone)
65             </td>
66         </tr>
67     </table>
68     <br />
69     Page @(Model.PageCount < Model.PageNumber ? 0 : Model.PageNumber) of @Model.PageCount
70
71     @Html.PagedListPager(Model, page => Url.Action("ListUtilisateur",
72         new { page, searchString = ViewBag.searchString }))
73 </div>
```

Sur le bas de page nous allons ajouter quelques lignes d'instructions :

- Une pour donner la page courante sur le nombre total de page
- Une pour donner le numéro des pages

```
<br />
Page @(Model.PageCount < Model.PageNumber ? 0 : Model.PageNumber) of @Model.PageCount

@Html.PagedListPager(Model, page => Url.Action("Index",
    new { page }))
```

# MVC ASP.NET

The screenshot shows a browser window for 'Index - My ASP.NET Appl...' at 'localhost:56787/Tuteurs'. The page title is 'Index' and the sub-page title is 'Create New'. A table lists two entries:

CNI/Passport	Nom	Prenom	Adresse	Téléphone	Email	Civilité	Parenté
THIOYE	Matar	Rufisque		00221778906767	papisthiuye@gmail.com	M.	Pere
SENE	DIATTA	Thiaroye		00221764532121	diatta@sene.sn	M.	Pere

Below the table, it says 'Page 1 of 1' with a single page number '1' in a blue box. At the bottom, it says '© 2016 - My ASP.NET Application'.

## Recherche sur une page

The screenshot shows a header with 'ORBUS' and links for 'Accueil', 'Utilisateurs', 'À propos de', and 'Contact'. On the right, it says 'Bonjour fgueye!' and 'Se déconnecter'. Below is a section titled 'Liste des utilisateurs' with a 'Nouveau' button and a search bar. It includes a search input field and a 'Rechercher' button. A table lists three users:

	Email	Identifiant	Nom	Prénom	Adresse	Téléphone
	mthioye@gainde2000.sn	mthioye@gainde2000.sn	THIOYE	Matar	Rufisque Santa Yalla	775684064
	mathioye@gainde2000.sn	mthioye	THIOYE	Matar	Rufisque Santa Yalla	775684064
	fgueye@gainde2000.sn	fgueye	GUEYE	Fama	Golf NORD, guediawaye	775783245

Below the table, it says 'Page 1 of 1' with a single page number '1' in a blue box.

## Source View

```
<div id="divSearch" style="display:none;">
    @using (Html.BeginForm("ListUtilisateur", "Account", FormMethod.Get))
    {
        <p>
            @ResourceView.Identifiant: @Html.TextBox("SearchString", ViewBag.searchString
            as string, new { @class = "form-control" })
            <input type="submit" value="@ResourceView.Rechercher" class="glyphicon
            glyphicon-search btn btn-info" />
        </p>
    }
</div>
```

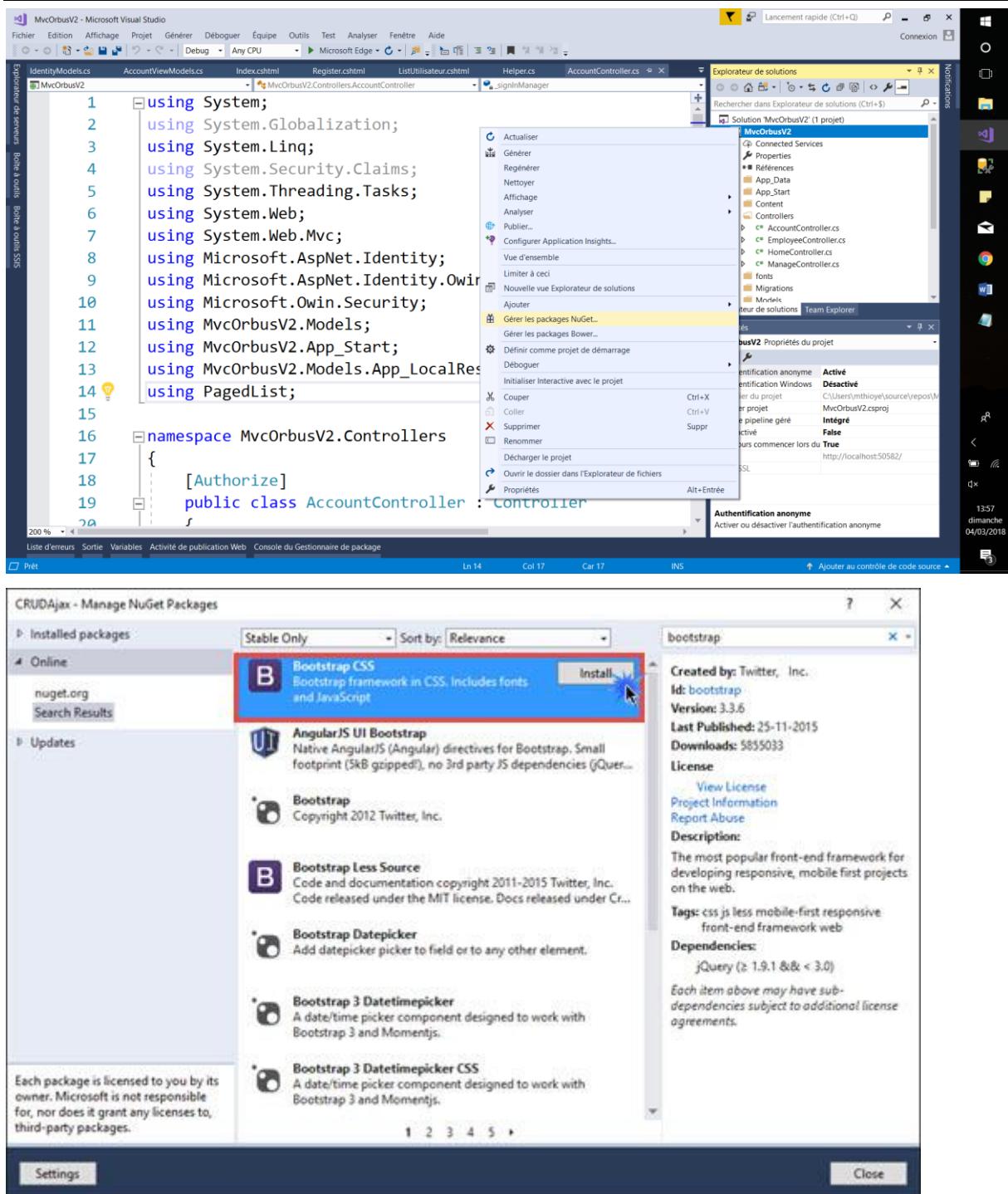
```
<script type="text/javascript">
    $(document).ready( function() {
        $('.flash-messages').delay(5000).fadeOut();
    });
    function ShowSearch() {
        var x = document.getElementById("divSearch");
        if (x.style.display === "none") {
            x.style.display = "block";
        } else {
            x.style.display = "none";
        }
    }
</script>
```

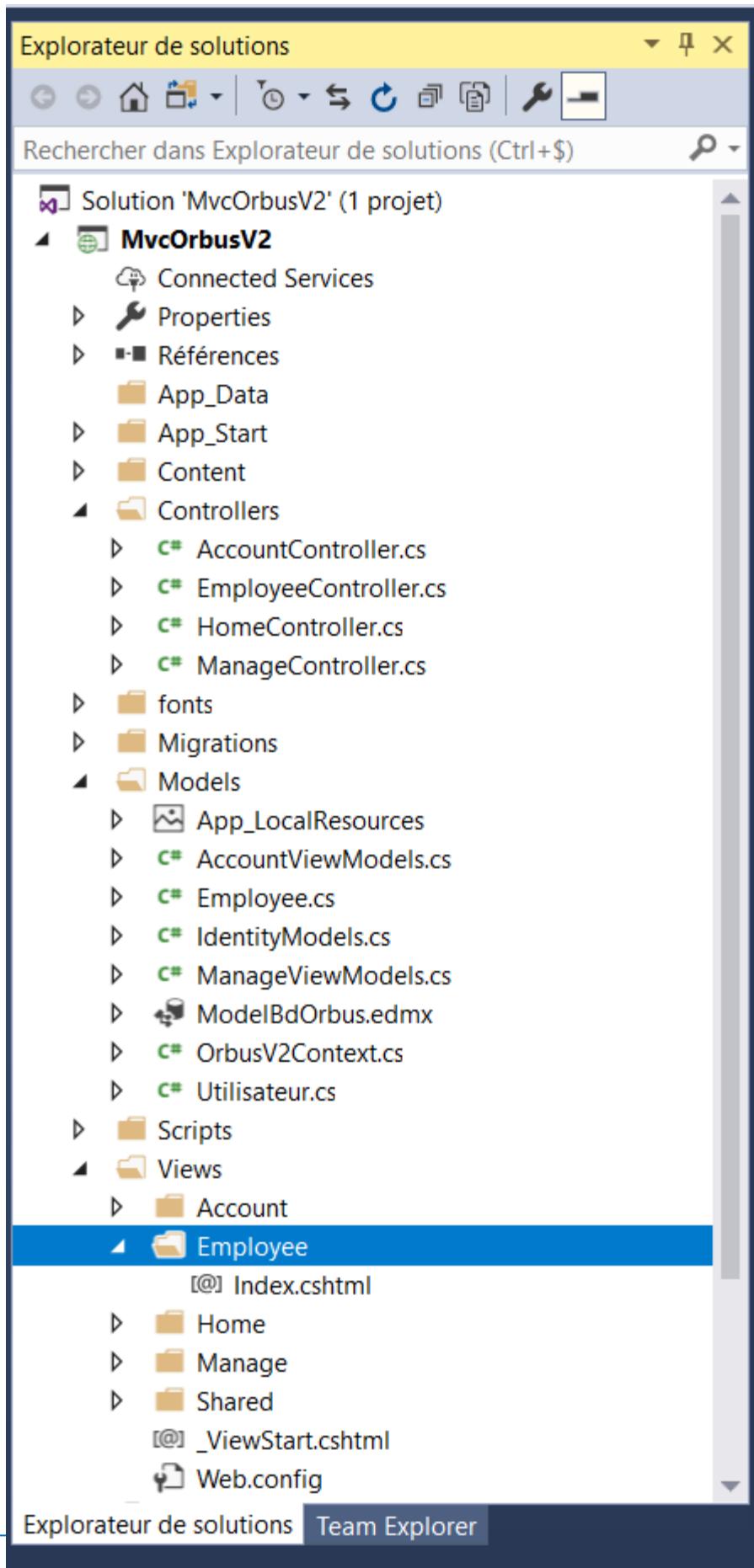
### AJAX

AJAX (JavaScript et XML asynchrones) est utilisé pour mettre à jour des parties de la page existante et pour récupérer les données du serveur de manière asynchrone. AJAX améliore les performances de l'application Web et rend l'application plus interactive.

Bootstrap est l'un des framework HTML, CSS et JS les plus populaires pour le développement de premiers projets mobiles et réactifs sur le Web.

# MVC ASP.NET





### Employee.cs Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace MvcOrbusV2.Models
{
    public class Employee
    {
        [Key]
        [ScaffoldColumn(false)]
        public int EmployeeID { get; set; }
        [Display(Name = "Nom prénom"), Required(ErrorMessage = "*")]
        public string Name { get; set; }
        [Display(Name = "Age"), Required(ErrorMessage = "*")]
        public int Age { get; set; }
        [Display(Name = "Etat"), Required(ErrorMessage = "*")]
        public string State { get; set; }
        [Display(Name = "Pays"), Required(ErrorMessage = "*")]
        public string Country { get; set; }
    }
}
```

### Controller vide

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcOrbusV2.Models;

namespace MvcOrbusV2.Controllers
{
    public class EmployeeController : Controller
    {
        OrbusV2Context db = new OrbusV2Context();
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
        public JsonResult List()
        {
            return Json(db.Employee.ToList(), JsonRequestBehavior.AllowGet);
        }
        public JsonResult Add(Employee emp)
        {
            db.Employee.Add(emp);
            db.SaveChanges();
            return Json(1, JsonRequestBehavior.AllowGet);
        }
}
```

```
        }
        public JsonResult GetbyID(int ID)
        {
            var Employee = db.Employee.ToList().Find(x => x.EmployeeID.Equals(ID));
            return Json(Employee, JsonRequestBehavior.AllowGet);
        }
        public JsonResult Update(Employee emp)
        {
            Employee e = db.Employee.Find(emp.EmployeeID);
            e.Age = emp.Age;
            e.Country = emp.Country;
            e.State = emp.State;
            e.Name = emp.Name;
            db.SaveChanges();
            return Json(1, JsonRequestBehavior.AllowGet);
        }
        public JsonResult Delete(int ID)
        {
            Employee e = db.Employee.Find(ID);
            db.Employee.Remove(e);
            db.SaveChanges();
            return Json(0, JsonRequestBehavior.AllowGet);
        }
    }
}
```

### View Employee

```
<div class="container">
    <h2>Employees Record</h2>
    <button type="button" class="btn btn-primary" data-toggle="modal" data-target="#myModal" onclick="clearTextBox();">Add New Employee</button><br /><br />
    <table class="table table-bordered table-hover">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Age</th>
                <th>State</th>
                <th>Country</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody class="tbody"></tbody>
    </table>
</div>
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModallLabel" aria-hidden="true">
    <div class="modal-dialog">
```

```

<div class="modal-content">
    <div class="modal-header">
        @*<button type="button" class="close" data-dismiss="modal"><span aria-hidden="true">x</span></button>*@
        <button type="button" class="close" data-dismiss="modal"></button>
        <h4 class="modal-title" id="myModalLabel">Add Employee</h4>
    </div>
    <div class="modal-body">
        <form>
            <div class="form-group">
                <label for="EmployeeId">ID</label>
                <input type="text" class="form-control" id="EmployeeID" placeholder="Id" disabled="disabled" />
            </div>
            <div class="form-group">
                <label for="Name">Name</label>
                <input type="text" class="form-control" id="Name" placeholder="Name" />
            </div>
            <div class="form-group">
                <label for="Age">Age</label>
                <input type="text" class="form-control" id="Age" placeholder="Age" />
            </div>
            <div class="form-group">
                <label for="State">State</label>
                <input type="text" class="form-control" id="State" placeholder="State" />
            </div>
            <div class="form-group">
                <label for="Country">Country</label>
                <input type="text" class="form-control" id="Country" placeholder="Country" />
            </div>
        </form>
    </div>
    <div class="modal-footer">
        <button type="button" class="btn btn-primary" id="btnAdd" onclick="return Add();">Add</button>
        <button type="button" class="btn btn-primary" id="btnUpdate" style="display:none;" onclick="Update();">Update</button>
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
    </div>
</div>

```

### Javascript Employee

```

//Load Data in Table when documents is ready
$(document).ready(function () {
    loadData();
});
//Load Data function
function loadData() {
    $.ajax({
        url: "/Employee/List",
        type: "GET",
        contentType: "application/json; charset=utf-8",
        dataType: "json",

```

```

success: function (result) {
    var html = "";
    $.each(result, function (key, item) {
        html += '<tr>';
        html += '<td>' + item.EmployeeID + '</td>';
        html += '<td>' + item.Name + '</td>';
        html += '<td>' + item.Age + '</td>';
        html += '<td>' + item.State + '</td>';
        html += '<td>' + item.Country + '</td>';
        html += '<td><a href="#" onclick="return getbyID(' + item.EmployeeID +
')">Edit</a> | <a href="#" onclick="Delete(' + item.EmployeeID + ')">Delete</a></td>';
        html += '</tr>';
    });
    $('.tbody').html(html);
},
error: function (errormessage) {
    alert(errormessage.responseText);
}
});
}

//Add Data Function
function Add() {
    var res = validate();
    if (res == false) {
        return false;
    }
    var empObj = {
        EmployeeID: $('#EmployeeID').val(),
        Name: $('#Name').val(),
        Age: $('#Age').val(),
        State: $('#State').val(),
        Country: $('#Country').val()
    };
    $.ajax({
        url: "/Employee/Add",
        data: JSON.stringify(empObj),
        type: "POST",
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        success: function (result) {
            loadData();
            $('#myModal').modal('hide');
        },
        error: function (errormessage) {
            alert(errormessage.responseText);
        }
    });
}

//Function for getting the Data Based upon Employee ID
function getbyID(EmpID) {
    $('#Name').css('border-color', 'lightgrey');
    $('#Age').css('border-color', 'lightgrey');
    $('#State').css('border-color', 'lightgrey');
    $('#Country').css('border-color', 'lightgrey');
    $.ajax({
        url: "/Employee/getbyID/" + EmpID,
        type: "GET",
        contentType: "application/json; charset=UTF-8",
        dataType: "json",
        success: function (result) {
}

```

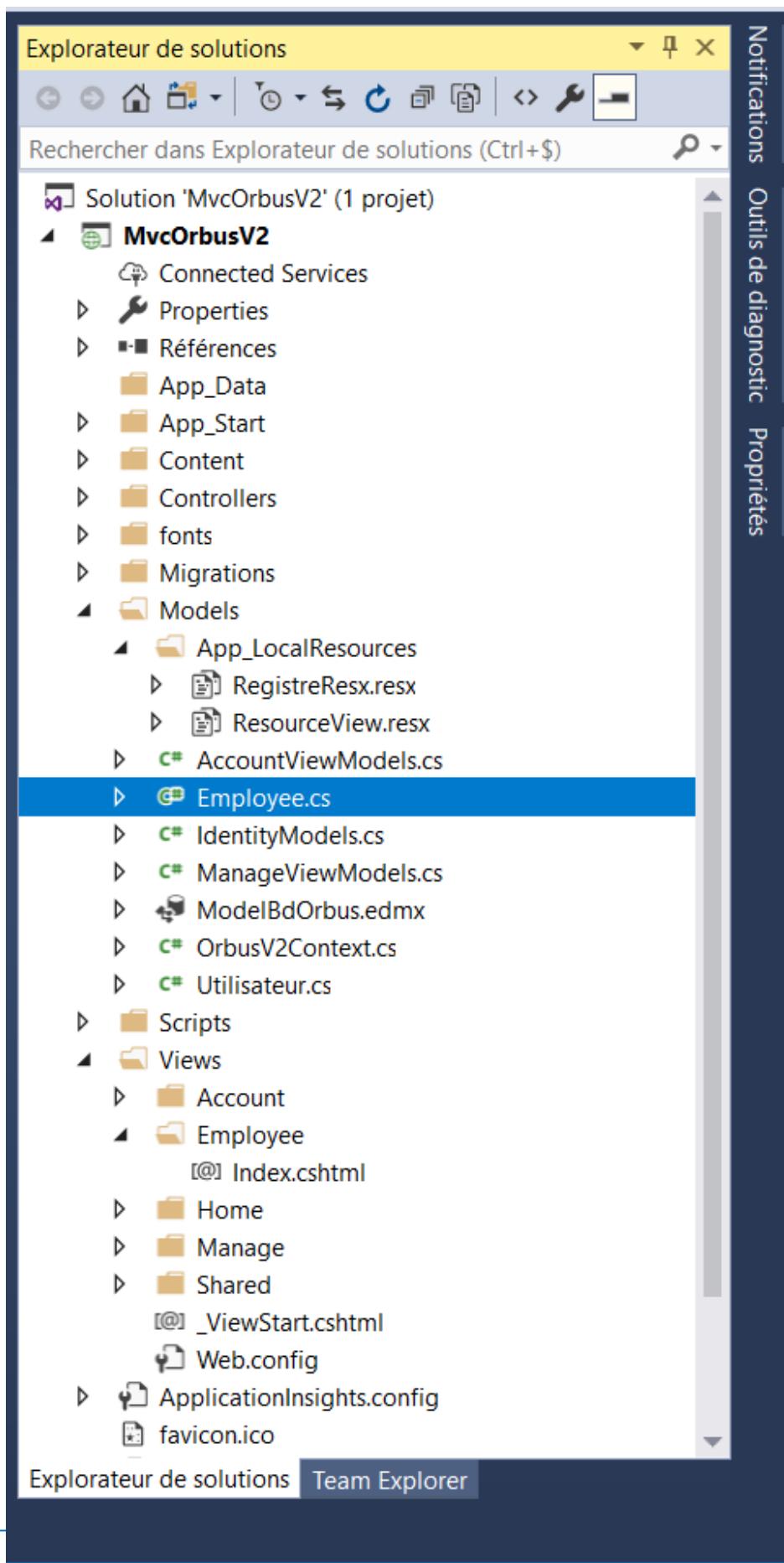
```
$('#EmployeeID').val(result.EmployeeID);
$('#Name').val(result.Name);
$('#Age').val(result.Age);
$('#State').val(result.State);
$('#Country').val(result.Country);
$('#myModal').modal('show');
$('#btnUpdate').show();
$('#btnAdd').hide();
},
error: function (errormessage) {
    alert(errormessage.responseText);
}
});
return false;
}
//function for updating employee's record
function Update() {
    var res = validate();
    if (res == false) {
        return false;
    }
    var empObj = {
        EmployeeID: $('#EmployeeID').val(),
        Name: $('#Name').val(),
        Age: $('#Age').val(),
        State: $('#State').val(),
        Country: $('#Country').val(),
    };
    $.ajax({
        url: "/Employee/Update",
        data: JSON.stringify(empObj),
        type: "POST",
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        success: function (result) {
            loadData();
            $('#myModal').modal('hide');
            $('#EmployeeID').val("");
            $('#Name').val("");
            $('#Age').val("");
            $('#State').val("");
            $('#Country').val("");
        },
        error: function (errormessage) {
            alert(errormessage.responseText);
        }
    });
}
//function for deleting employee's record
function Delete(ID) {
    var ans = confirm("Are you sure you want to delete this Record?");
    if (ans) {
        $.ajax({
            url: "/Employee/Delete/" + ID,
            type: "POST",
            contentType: "application/json; charset=UTF-8",
            dataType: "json",
            success: function (result) {
                loadData();
            },
        });
    }
}
```

```
        error: function (errormessage) {
            alert(errormessage.responseText);
        }
    });
}
//Function for clearing the textboxes
function clearTextBox() {
    $('#EmployeeID').val("");
    $('#Name').val("");
    $('#Age').val("");
    $('#State').val("");
    $('#Country').val("");
    $('#btnUpdate').hide();
    $('#btnAdd').show();
    $('#Name').css('border-color', 'lightgrey');
    $('#Age').css('border-color', 'lightgrey');
    $('#State').css('border-color', 'lightgrey');
    $('#Country').css('border-color', 'lightgrey');
}
//Validation using jquery
function validate() {
    var isValid = true;
    if ($('#Name').val().trim() == "") {
        $('#Name').css('border-color', 'Red');
        isValid = false;
    }
    else {
        $('#Name').css('border-color', 'lightgrey');
    }
    if ($('#Age').val().trim() == "") {
        $('#Age').css('border-color', 'Red');
        isValid = false;
    }
    else {
        $('#Age').css('border-color', 'lightgrey');
    }
    if ($('#State').val().trim() == "") {
        $('#State').css('border-color', 'Red');
        isValid = false;
    }
    else {
        $('#State').css('border-color', 'lightgrey');
    }
    if ($('#Country').val().trim() == "") {
        $('#Country').css('border-color', 'Red');
        isValid = false;
    }
    else {
        $('#Country').css('border-color', 'lightgrey');
    }
    return isValid;
}
```

### Dynamic Load Data

### Fichier de ressource

## MVC ASP.NET



## MVC ASP.NET

The screenshot shows the Visual Studio Resource Editor interface. The title bar says "MVC ASP.NET". The tab bar includes "ResourceView.resx", "EmployeeController.cs", "Employee.cs", "Index.cshtml", "Register.cshtml", "OrbusV2Context.cs", and "RegistreResx.resx". The "RegistreResx.resx" tab is selected. The toolbar has buttons for "Chaines", "Ajouter une ressource", "Supprimer une ressource", and "Modificateur d'accès: Public". The main area is a table with columns "Nom", "Valeur", and "Commentaire". The table contains the following data:

Nom	Valeur	Commentaire
Adresse	Adresse	
ConfirmPassword	Confirmer le mot de passe	
Email	Courrier électronique	
Nom	Nom	
Password	Mot de passe	
Prenom	Prénom	
RememberMe	Mémoriser le mot de passe ?	
Tel	Téléphone	
UserName	Nom utilisateur	
*		

# MVC ASP.NET

Screenshot of the Visual Studio IDE showing two code editors side-by-side.

The top editor displays the `Login.cshtml` file:

```

1  @using MvcOrbusV2.Models
2  @using MvcOrbusV2.Models.App_LocalResources
3  @model LoginViewModel
4  @{
5      ViewBag.Title = @ResourceView.Connexion;
6  }
7
8  <h2>@ViewBag.Title.</h2>
9  <div class="row">
10    <div class="col-md-8">
11      <section id="loginForm">
12          <using(Html.BeginForm("Login", "Account", new { ReturnUrl = ViewBag.ReturnUrl }, FormMethod.Post, new { @class = "form-horizontal" })>
13          {
14              @Html.AntiForgeryToken()
15              <h4>@ResourceView.CompteLocal</h4>
16              <br />
17              @Html.ValidationSummary(true, "", new { @class = "text-danger" })
18              <div class="form-group">
19                  @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
20                  <div class="col-md-10">
21                      @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
22                      @Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-danger" })
23                  </div>
24              </div>
25              <div class="form-group">
26                  @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
27                  <div class="col-md-10">
28                      @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
29                      @Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })
30                  </div>
31              </div>
32              <div class="form-group">
33                  <div class="col-md-offset-2 col-md-10">
34                      <div class="checkbox">
35                          @Html.CheckBoxFor(m => m.RememberMe)
36                          @Html.LabelFor(m => m.RememberMe)
37                      </div>
38                  </div>
39              </div>
40          </div>
41      </section>
42  </div>
43  </div>
44
45  <script src="~/Scripts/jquery-1.10.2.min.js"></script>
46  <script src="~/Scripts/jquery.validate.min.js"></script>
47  <script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87

```

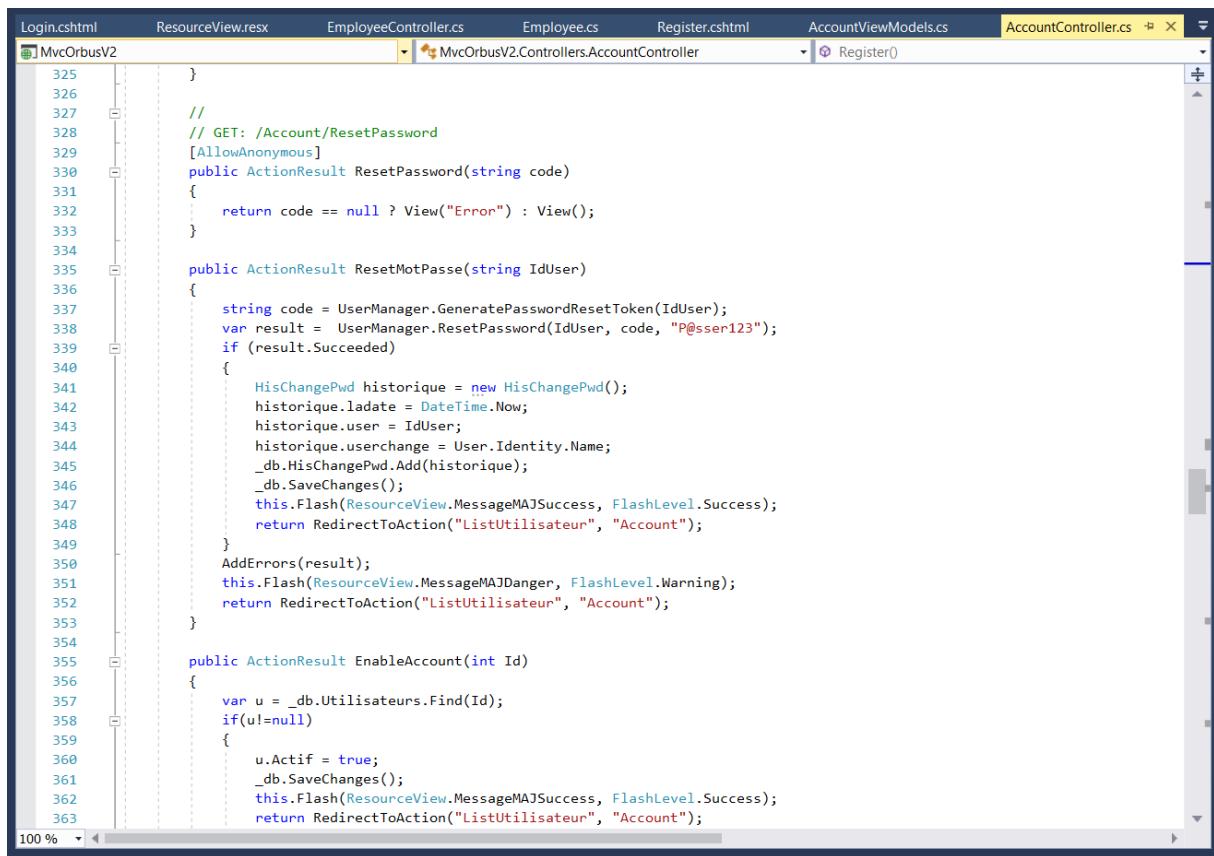
The bottom editor displays the `AccountViewModels.cs` file:

```

49
50  public class LoginViewModel
51  {
52      [Required]
53      [Display(Name = "Email", ResourceType = typeof(RegistreResx))]
54      /*[EmailAddress]*/
55      public string Email { get; set; }
56
57      [Required]
58      [DataType(DataType.Password)]
59      [Display(Name = "Password", ResourceType = typeof(RegistreResx))]
60      public string Password { get; set; }
61
62      [Display(Name = "RememberMe", ResourceType = typeof(RegistreResx))]
63      public bool RememberMe { get; set; }
64  }
65
66  public class RegisterViewModel
67  {
68      [Required]
69      [EmailAddress]
70      [Display(Name = "Email", ResourceType = typeof(RegistreResx))]
71      public string Email { get; set; }
72
73      [Required]
74      [MaxLength(80, ErrorMessage = "la taille maximale est 80 caractères")]
75      [Display(Name = "UserName", ResourceType = typeof(RegistreResx))]
76      public string UserName { get; set; }
77
78      [Required]
79      [MaxLength(80, ErrorMessage = "la taille maximale est 80 caractères")]
80      [Display(Name = "Nom", ResourceType = typeof(RegistreResx))]
81      public string Nom { get; set; }
82
83      [Required]
84      [MaxLength(80, ErrorMessage = "la taille maximale est 80 caractères")]
85      [Display(Name = "Prenom", ResourceType = typeof(RegistreResx))]
86      public string Prenom { get; set; }
87

```

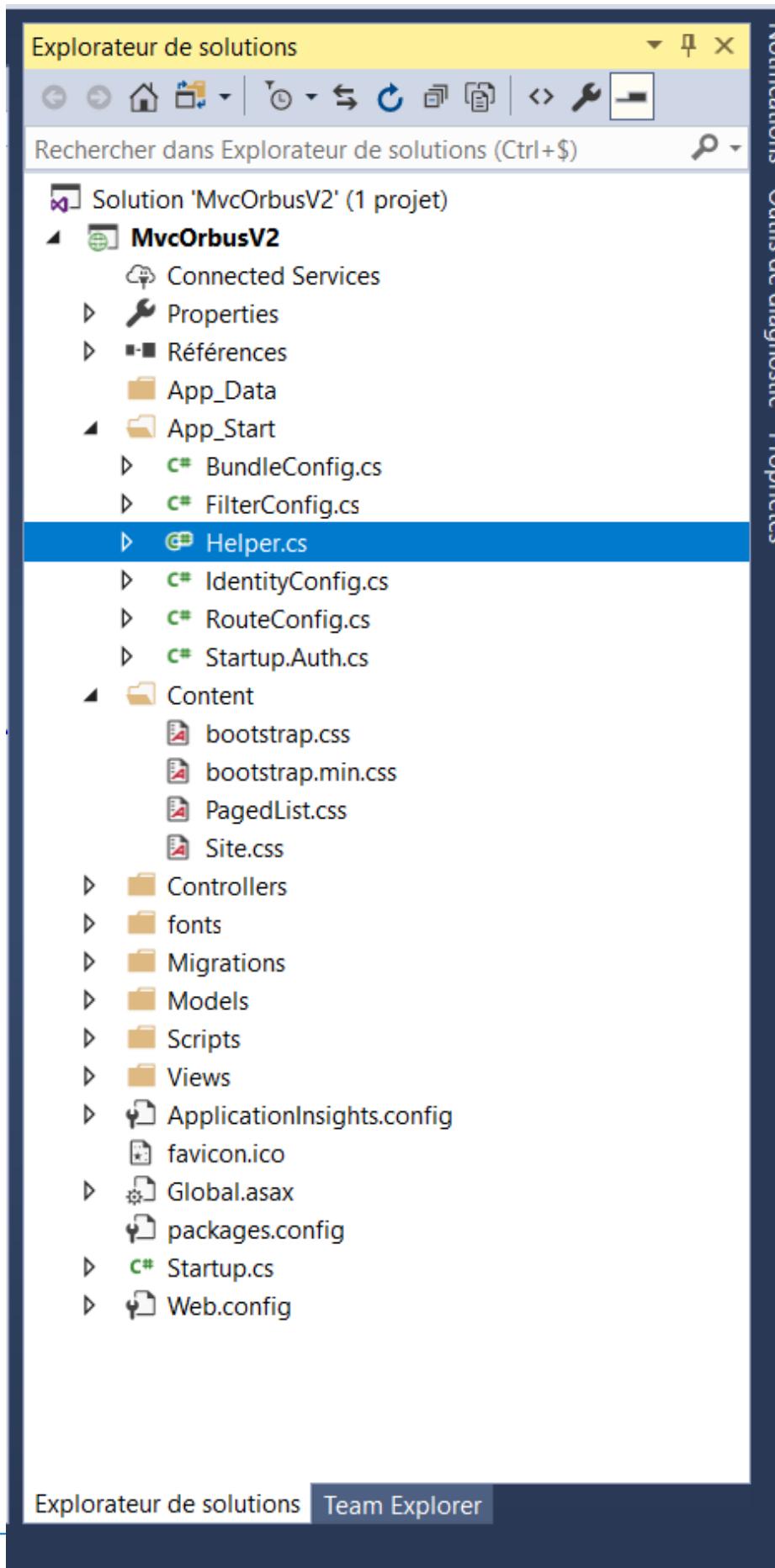
## MVC ASP.NET



```
325 }
326 }
327 }
328 // GET: /Account/ResetPassword
329 [AllowAnonymous]
330 public ActionResult ResetPassword(string code)
331 {
332     if (code == null) return View("Error");
333 }
334
335 public ActionResult ResetMotPasse(string IdUser)
336 {
337     string code = UserManager.GeneratePasswordResetToken(IdUser);
338     var result = UserManager.ResetPassword(IdUser, code, "P@sser123");
339     if (result.Succeeded)
340     {
341         HisChangePwd historique = new HisChangePwd();
342         historique.ladate = DateTime.Now;
343         historique.user = IdUser;
344         historique.userchange = User.Identity.Name;
345         _db.HisChangePwd.Add(historique);
346         _db.SaveChanges();
347         this.Flash(ResourceView.MessageMAJSuccess, FlashLevel.Success);
348         return RedirectToAction("ListUtilisateur", "Account");
349     }
350     AddErrors(result);
351     this.Flash(ResourceView.MessageMAJDanger, FlashLevel.Warning);
352     return RedirectToAction("ListUtilisateur", "Account");
353 }
354
355 public ActionResult EnableAccount(int Id)
356 {
357     var u = _db.Utilisateurs.Find(Id);
358     if (u != null)
359     {
360         u.Actif = true;
361         _db.SaveChanges();
362         this.Flash(ResourceView.MessageMAJSuccess, FlashLevel.Success);
363         return RedirectToAction("ListUtilisateur", "Account");
364     }
365 }
```

Notification provenant du Controller

## MVC ASP.NET



## MVC ASP.NET

The screenshot shows two code editor windows side-by-side in a Microsoft Visual Studio interface.

**Top Window (Helper.cs):**

```
28     client.Send(message);
29 }
30 }
31 }
32 }
33 }
34 }
35 }
36 }
37
38 public enum FlashLevel
39 {
40     Info = 1,
41     Success = 2,
42     Warning = 3,
43     Danger = 4
44 }
45
46 public static class FlashHelper
47 {
48     public static void Flash(this Controller controller, string message, FlashLevel level)
49     {
50         IList<string> messages = null;
51         string key = String.Format("flash-{0}", level.ToString().ToLower());
52
53         messages = (controller.TempData.ContainsKey(key))
54             ? (IList<string>)controller.TempData[key]
55             : new List<string>();
56
57         messages.Add(message);
58
59         controller.TempData[key] = messages;
60     }
61 }
```

**Bottom Window (AccountController.cs):**

```
325 }
326
327 }
328 // GET: /Account/ResetPassword
329 [AllowAnonymous]
330 public ActionResult ResetPassword(string code)
331 {
332     return code == null ? View("Error") : View();
333 }
334
335
336 public ActionResult ResetMotPasse(string IdUser)
337 {
338     string code = UserManager.GeneratePasswordResetToken(IdUser);
339     var result = UserManager.ResetPassword(IdUser, code, "P@sser123");
340     if (result.Succeeded)
341     {
342         HisChangePwd historique = new HisChangePwd();
343         historique.ladate = DateTime.Now;
344         historique.user = IdUser;
345         historique.userchange = User.Identity.Name;
346         _db.HisChangePwd.Add(historique);
347         _db.SaveChanges();
348         this.Flash(ResourceView.MessageMAJSuccess, FlashLevel.Success);
349         return RedirectToAction("ListUtilisateur", "Account");
350     }
351     AddErrors(result);
352     this.Flash(ResourceView.MessageMJDanger, FlashLevel.Warning);
353     return RedirectToAction("ListUtilisateur", "Account");
354 }
355
356 public ActionResult EnableAccount(int Id)
357 {
358     var u = _db.Utilisateurs.Find(Id);
359     if(u!=null)
360     {
361         u.Actif = true;
362         _db.SaveChanges();
363         this.Flash(ResourceView.MessageMAJSuccess, FlashLevel.Success);
364         return RedirectToAction("ListUtilisateur", "Account");
365     }
366 }
```

## MVC ASP.NET

The screenshot shows a code editor with two tabs open: `_Flash.cshtml` and `_Layout.cshtml`.

`_Flash.cshtml` (Top Tab):

```
1  @using MvcOrbusV2.App_Start
2  @helper FlashMessage(System.Web.Mvc.TempDataDictionary tempData)
3  {
4      <div class="flash-messages">
5          @foreach (FlashLevel level in (FlashLevel[])Enum.GetValues(typeof(FlashLevel)))
6          {
7              string type = level.ToString().ToLower();
8              string key = "flash-" + type;
9
10             if (tempData.ContainsKey(key))
11             {
12                 IList<string> messages = (IList<string>)tempData[key];
13
14                 foreach (string message in messages)
15                 {
16                     <p class="alert alert-@type" role="alert">@message</p>
17                 }
18             }
19         }
20     </div>
21 }
22
23 @FlashMessage(TempData)
```

`_Layout.cshtml` (Bottom Tab):

```
100 % ↻
25     <ul class="nav navbar-nav">
26         <li>@Html.ActionLink(@ResourceView.Accueil, "Index", "Home")</li>
27         <li>@Html.ActionLink(@ResourceView.Utilisateurs, "ListUtilisateur", "Account")</li>
28         <li>@Html.ActionLink(@ResourceView.Employee, "Index", "Employee")</li>
29         <li>@Html.ActionLink(@ResourceView.Propos, "About", "Home")</li>
30         <li>@Html.ActionLink(@ResourceView.Contact, "Contact", "Home")</li>
31     </ul>
32     @Html.Partial("_LoginPartial")
33   </div>
34 </div>
35 <div class="container body-content">
36     @Html.Partial("_Flash")
37     @RenderBody()
38     <hr />
39     <footer>
40         <p>&copy; @DateTime.Now.Year - @ResourceView.Application</p>
41     </footer>
42 </div>
43
44 @Scripts.Render("~/bundles/jquery")
45 @Scripts.Render("~/bundles/bootstrap")
46 @RenderSection("scripts", required: false)
47 <script type="text/javascript">
48     $(document).ready( function() {
49         $('.flash-messages').delay(5000).fadeOut();
50     });
51     function ShowSearch() {
52         var x = document.getElementById("divSearch");
53         if (x.style.display === "none") {
54             x.style.display = "block";
55         } else {
56             x.style.display = "none";
57         }
58     }
59 </script>
60 </body>
61 </html>
```

# MVC ASP.NET

The screenshot shows two code editors in a Microsoft Visual Studio interface.

**Top Editor (Helper.cs):**

```
7  namespace MvcOrbusV2.App_Start
8  {
9      public class Helper
10     {
11         public static void SendMail(string address, string subject, string body)
12         {
13             using (MailMessage message = new MailMessage())
14             {
15                 message.Subject = subject;
16                 message.Body = "<pre>" + body + "</pre>";
17                 message.To.Add(address);
18                 message.Priority = MailPriority.High;
19                 message.IsBodyHtml = true;
20                 using (SmtpClient client = new SmtpClient())
21                 {
22                     try
23                     {
24                         client.Send(message);
25                     }
26                     catch (Exception e) { e.GetBaseException(); }
27                 }
28             }
29         }
30     }
31 }
32 }
```

**Bottom Editor (Web.config):**

```
7 <configSections>
8   <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkId=237468 -->
9   <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0
10  </configSections>
11 <connectionStrings>
12   <add name="OrbusV2Conn" connectionString="Data Source=DESKTOP-Q76I0Q2;Initial Catalog=bdOrbusV2;Integrated Security=True" provider
13   <add name="bdOrbusV2Entities" connectionString="metadata=res://*/Models.ModelBdOrbus.csdl|res://*/Models.ModelBdOrbus.ssdl|res://*/
14 <appSettings>
15   <add key="webpages:Version" value="3.0.0.0" />
16   <add key="webpages:Enabled" value="false" />
17   <add key="ClientValidationEnabled" value="true" />
18   <add key="UnobtrusiveJavaScriptEnabled" value="true" />
19 </appSettings>
20 <system.web>
21   <authentication mode="None" />
22   <compilation debug="true" targetFramework="4.6" />
23   <httpRuntime targetFramework="4.6" />
24   <httpModules>
25     <add name="ApplicationInsightsWebTracking" type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule, Microsoft.AI.
26   </httpModules>
27 </system.web>
28 <system.net>
29   <mailSettings>
30     <smtp from="papisthiuye@gmail.com">
31       <network host="smtp.gmail.com" port="465" enableSsl="true" password="*****" userName="papisthiuye@gmail.com" />
32     </smtp>
33   </mailSettings>
34   <defaultProxy enabled="true" />
35   <settings>
36     <!-- This setting causes .NET to check certificate revocation lists (CRL)
37         before trusting HTTPS certificates. But this setting tends to not
38         be allowed in shared hosting environments. -->
39     <!--<servicePointManager checkCertificateRevocationList="true"/>-->
40   </settings>
41 </system.net>
42 <system.webServer>
43   <modules>
44     <remove name="FormsAuthentication" />
45     <remove name="ApplicationInsightsWebTracking" />
46     <add name="ApplicationInsightsWebTracking" type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule, Microsoft.AI.
47   </modules>
48 </system.webServer>
```

### Gestion des erreurs

#### Impression

Pour l'impression nous allons utiliser crystal report. L'implémentation est composer de :

- La création de la classe ;
- La création du report ;
- La methode d'appel.

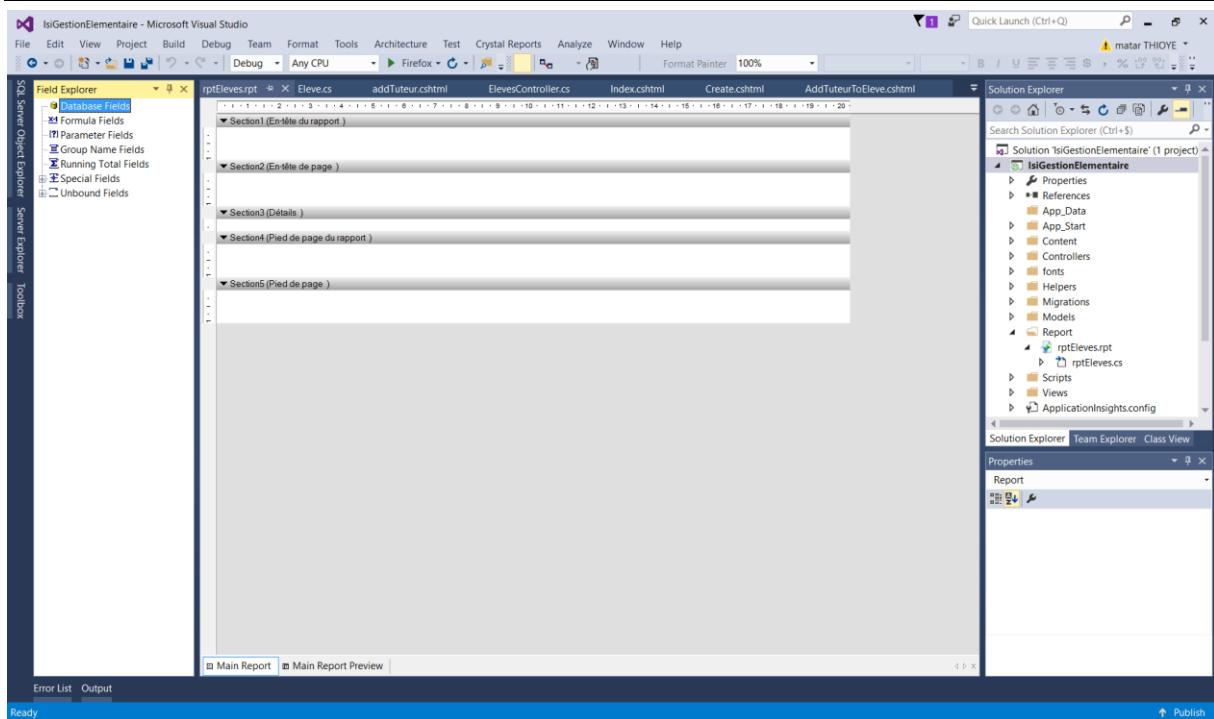
#### La création de la classe

```
public class rptEleve
{
    public string Matricule { get; set; }
    public string Nom { get; set; }
    public string Prenom { get; set; }
    public string Sexe { get; set; }
    public DateTime? DateNaiss { get; set; }
    public string LieuNaiss { get; set; }

}
```

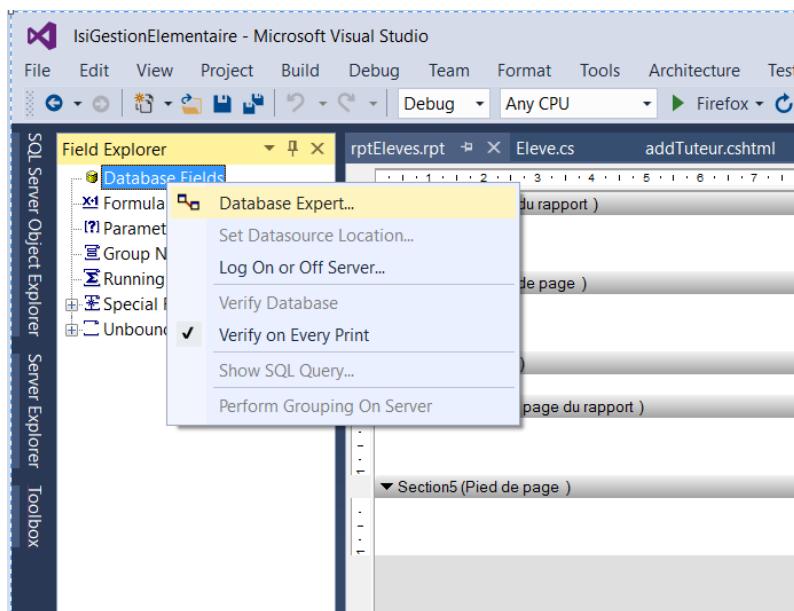
#### La création du rapport

## MVC ASP.NET



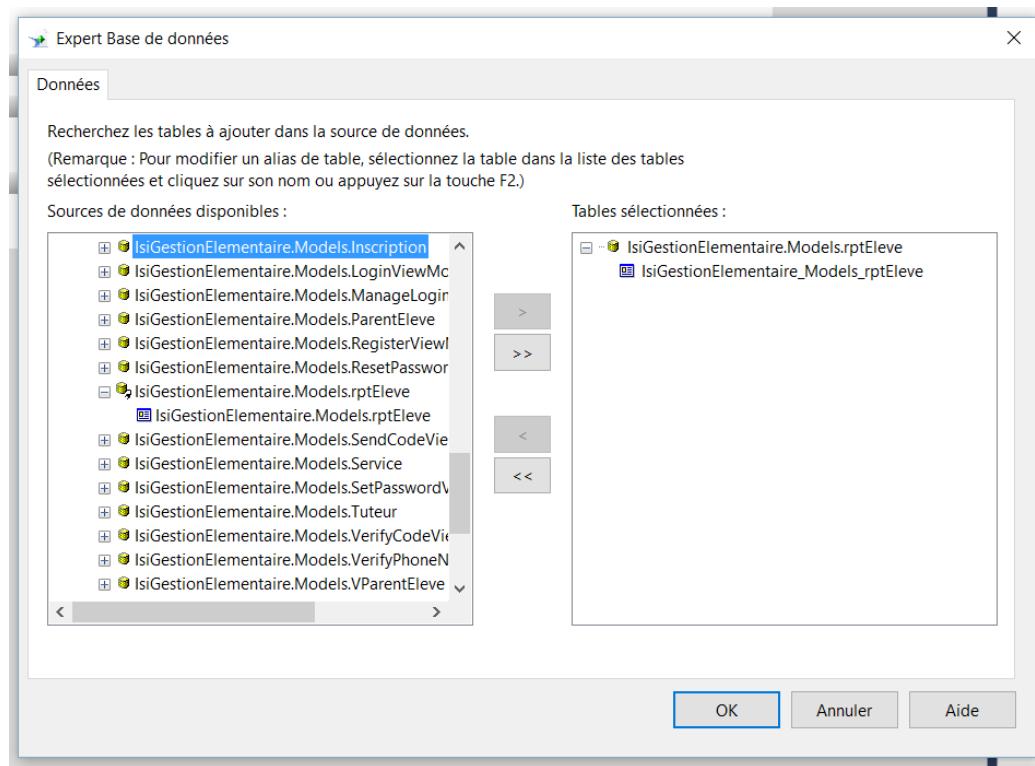
Le rapport est composé de :

- L'en-tête du rapport qui désigne la première page si le rapport fait plusieurs pages, sinon vous le fermer ;
- L'en-tête de page, comme son nom l'indique il sera répété sur toute les pages à la partie entête ;
- Détails, qui désigne les lignes d'informations ;
- Pied de page du rapport qui est la dernière page quand le rapport fait plusieurs pages, sinon vous le fermer ;
- Pied de page, répéter sur chaque page.

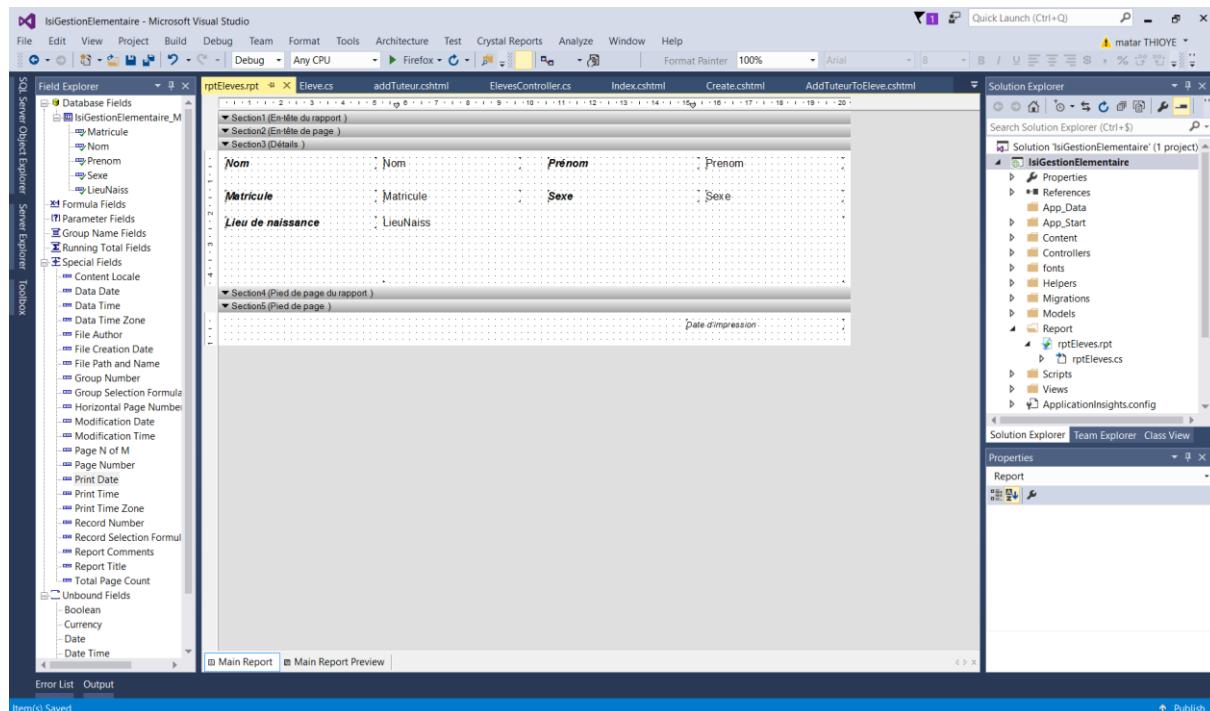


## MVC ASP.NET

Connecter votre rapport à votre class rptEleve et faites la mise en forme.



Bien faire la mise en forme



La méthode d'appel

```
public DataTable GetTableEleve(int id)
{
}
```

```
DataTable table = new DataTable();
table.Columns.Add("Matricule", typeof(string));
table.Columns.Add("Nom", typeof(string));
table.Columns.Add("Prenom", typeof(string));
table.Columns.Add("Sexe", typeof(string));
table.Columns.Add("DateNaiss", typeof(DateTime));
table.Columns.Add("LieuNaiss", typeof(string));
Eleve e = db.Eleve.Find(id);

table.Rows.Add(e.Matricule, e.Nom, e.Prenom, e.Sexe, e.DateNaiss,
e.LieuNaiss);

return table;
}

public ActionResult ReportEleve(int id)
{
    CrystalDecisions.CrystalReports.Engine.ReportDocument RptFraisInscription
= new CrystalDecisions.CrystalReports.Engine.ReportDocument();
    try
    {
        RptFraisInscription.Load(Server.MapPath("~/Report/rptEleve.rpt"));
        RptFraisInscription.SetDataSource(GetTableEleve(id));
        Stream stream =
RptFraisInscription.ExportToStream(CrystalDecisions.Shared.ExportFormatType.PortableDo
cFormat);
        Response.AppendHeader("Content-Disposition", "inline");
        return File(stream, "application/pdf");
    }
    finally
    {
        RptFraisInscription.Dispose();
        RptFraisInscription.Close();
    }
}
```

Appel de la méthode action au niveau de la vue.

```
@Html.NoEncodeActionLink("<span class='glyphicon glyphicon-print'></span>",
"Imprimer", "ReportEleve", "Eleve", routeValues: new { id = item.Id }, htmlAttributes:
new { data_modal = "", @class = "btn btn-default" })
```

## Index

Create New

Matricule	Nom	Prénom	sexe	Date de Naissance	Lieu de Naissance	
2016A	SARR	Omar	M	27/11/2012 00:00:00	Kaffrine	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>   
0001	Justice	la loi	M	12/12/2016 00:00:00	Dakar	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>   
0001	Justice	Omar	M	12/12/2016 00:00:00	Kaffrine	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>   
0001	thiam	dollar	M	12/12/2016 00:00:00	Kaffrine	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>   

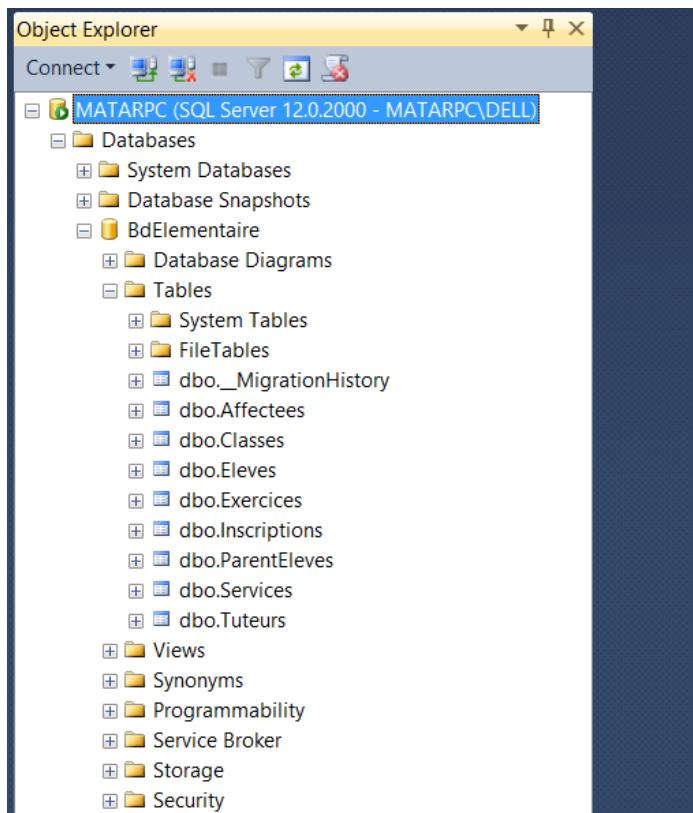
© 2016 - My ASP.NET Application

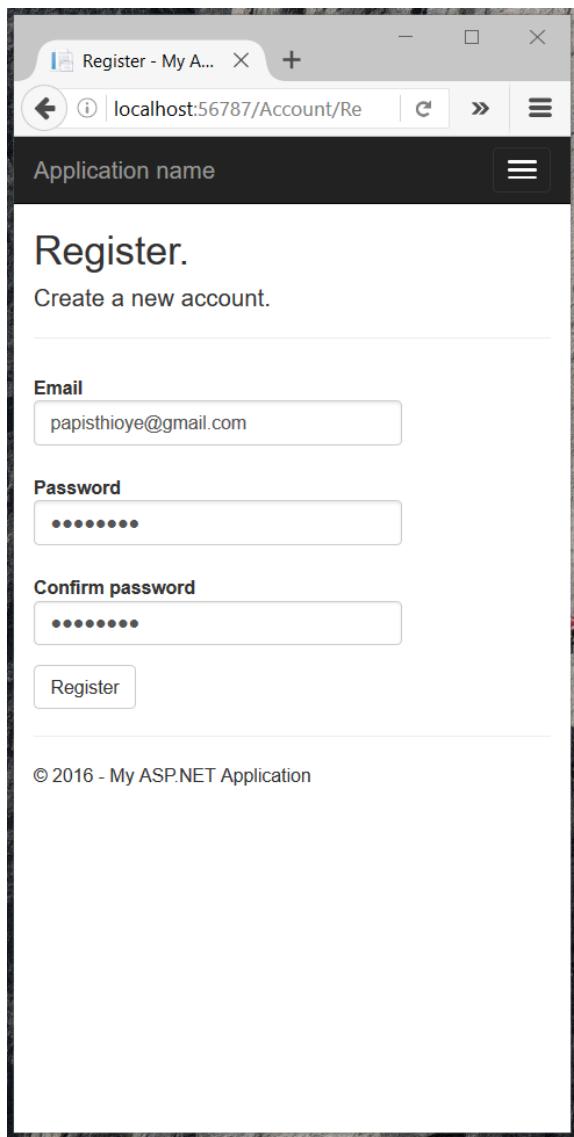


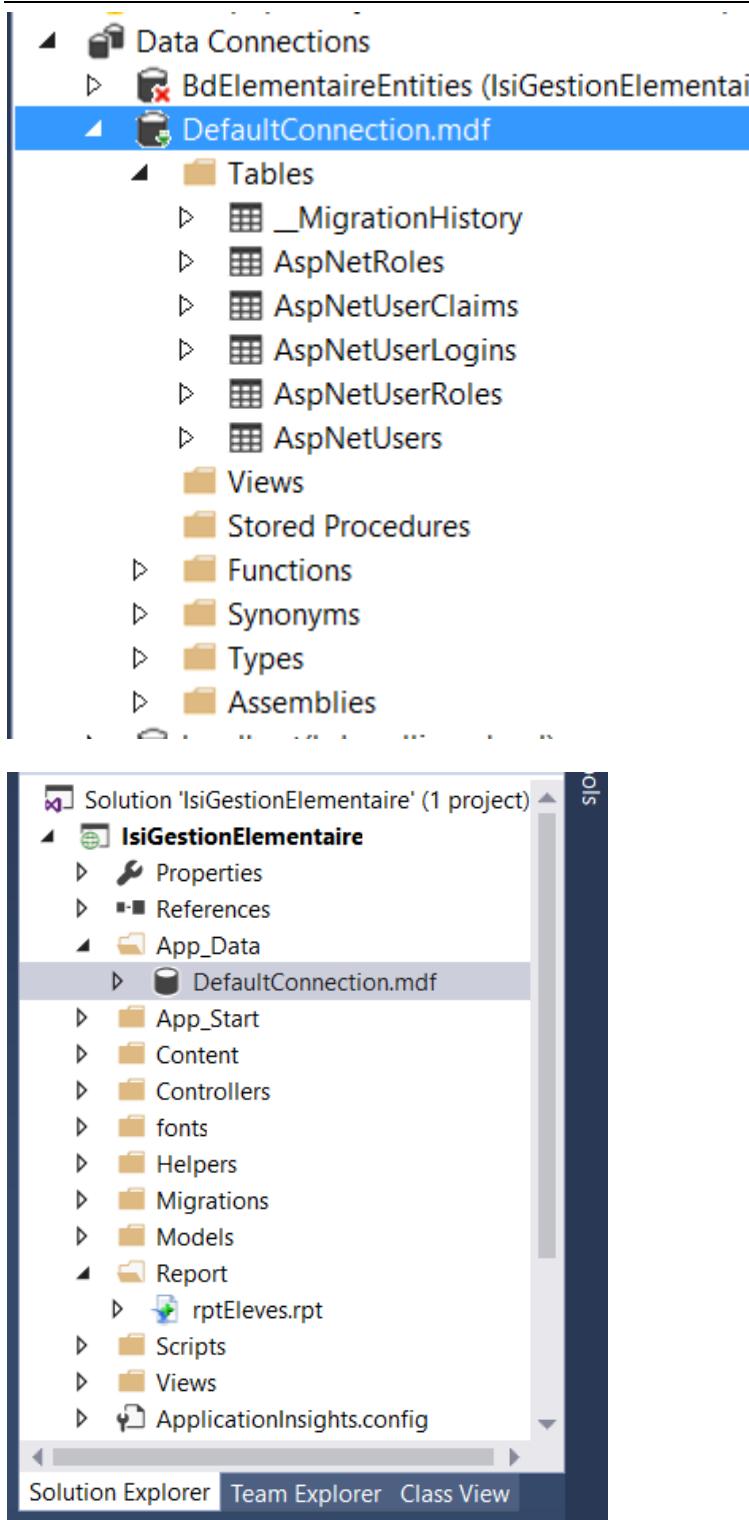
## Securite

Nous allons aborder l'authentification et les contrôles d'accès.

## MVC ASP.NET







### Authentication

In this article we will see how to use ASP.NET Identity in MVC Application for creating user roles and displaying the menu depending on user roles.

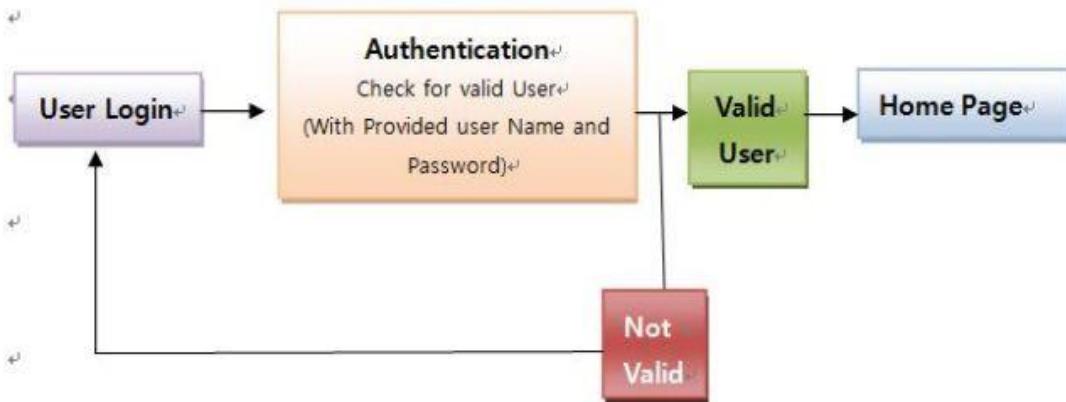
Here we will see how to:

- Create default admin role and other roles.
- Create default admin users.
- Add Username for new User Registration.
- Select User Role during User Registration.
- Change Login Email with User Name.
- Display Role Creation Menu only for Admin User.
- Display message for normal user.
- Redirect Unauthenticated users to default home page.

## Authentication and Authorization

### Authentication

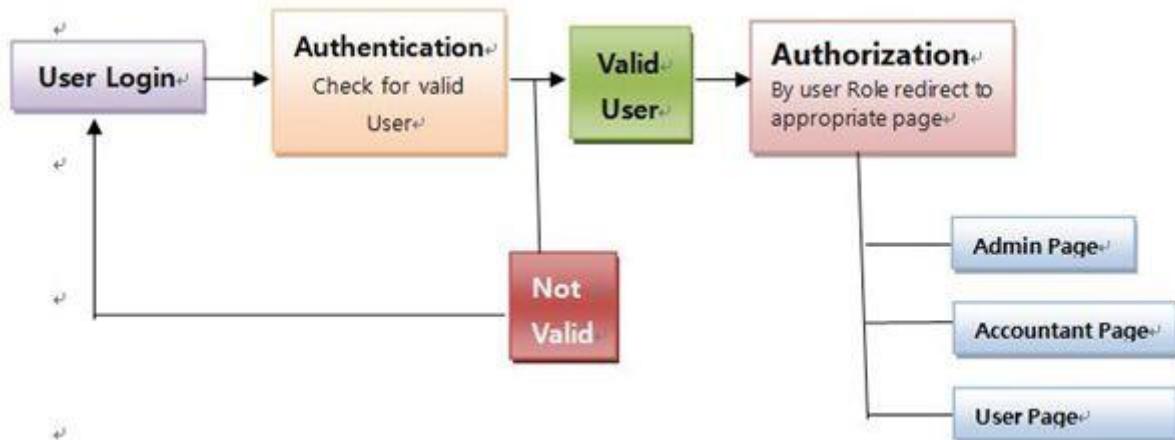
Check for the Valid User. Here the question is how to check whether a user is valid or not. When a user comes to a Web site for the first time he will register for that Web site. All his information, like user name, password, email, and so on will be stored in the Web site database. When a user enters his userID and password, the information will be checked with the database. If the user has entered the same userID and Password as in the database then he or she is a valid user and will be redirected to the Web site home page. If the user enters a UserID and/or Password that does not match the database then the login page will give a message, something like "Enter valid Name or Password". The entire process of checking whether the user is valid or not for accessing the Web site is called Authentication.



### Authorization

Once the user is authenticated he needs to be redirected to the appropriate page by his role. For example, when an Admin is logged in, then he is to be redirected to the Admin Page. If an Accountant is logged in, then he is to be redirected to his Accounts page. If an End User is logged in, then he is to

be redirected to his page.



## Building the Sample

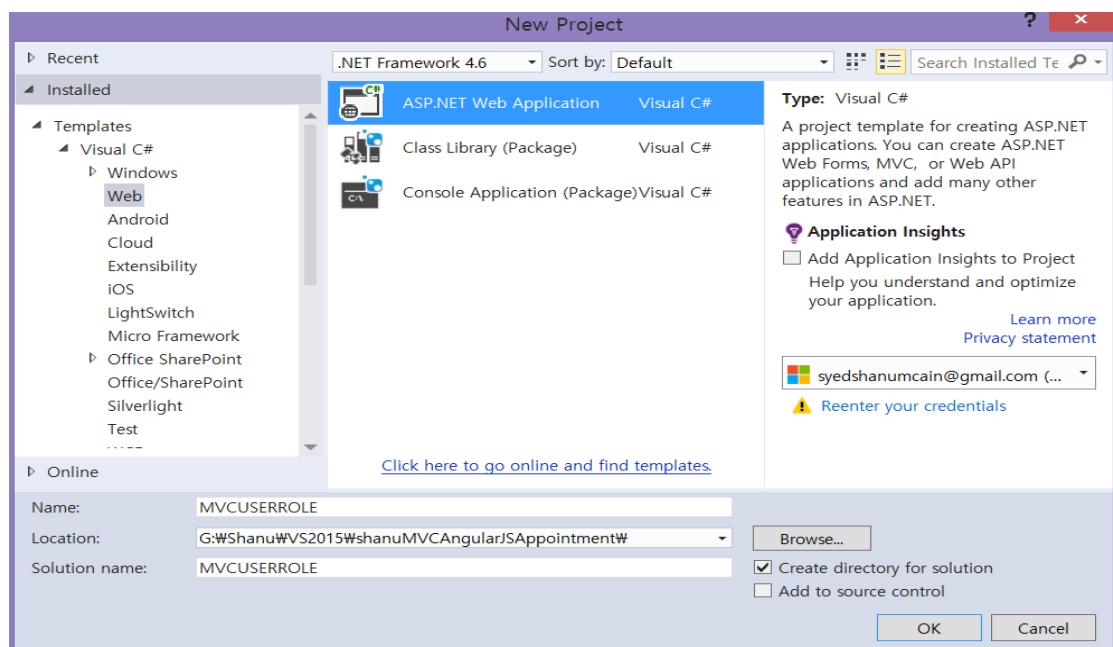
### Prerequisites

**Visual Studio 2015:** You can download it from [here](#).

## Description

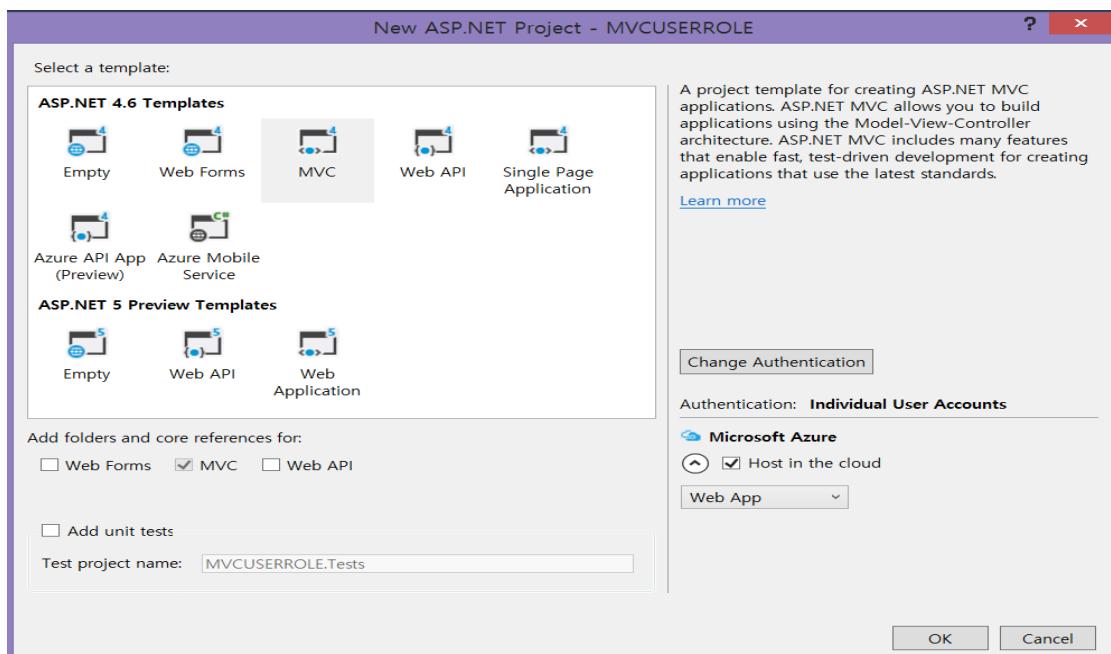
### Create your Web Application in Visual Studio 2015

After installing our Visual Studio 2015 click Start, then Programs and select **Visual Studio 2015 - ClickVisual Studio 2015**. Click New, then Project, select Web and then select **ASP.NET Web Application**. Enter your project name and click OK.



## MVC ASP.NET

Select MVC and click OK.



## Create a Database

Firstly, we will create a Database and set the connection string in **web.config** file for DefaultConnection with our new database connection. We will be using this database for ASP.NET Identity table creation and also our sample attendance Web project. Instead of using two databases as one for default ASP.NET user database and another for our Attendance DB, here we will be using one common database for both user details and for our sample web project demo.

Create Database: Run the following and create database script to create our sample test database.

SQL

```
-- =====
-- Author      : Shanu
-- Create date : 2016-01-17
-- Description : To Create Database

-- =====
--Script to create DB,Table and sample Insert data
USE MASTER;
-
- 1) Check for the Database Exists .If the database is exist then drop and
create new DB
IF EXISTS (SELECT [name] FROM sys.databases WHERE [name] = 'AttendanceDB' )

BEGIN
ALTER DATABASE AttendanceDB SET SINGLE_USER WITH ROLLBACK IMMEDIATE
DROP DATABASE AttendanceDB ;
```

END

```
CREATE DATABASE AttendanceDB  
GO  
USE AttendanceDB
```

### Web.Config

In web.config file we can find the DefaultConnection Connection string. By default ASP.NET MVC will use this connection string to create all ASP.NET Identity related tables like AspNetUsers, etc. For our application we also need to use database for other page activities instead of using two different databases, one for User details and one for our own functionality. Here I will be using one database where all ASP.NET Identity tables will be created and also we can create our own tables for other page uses.

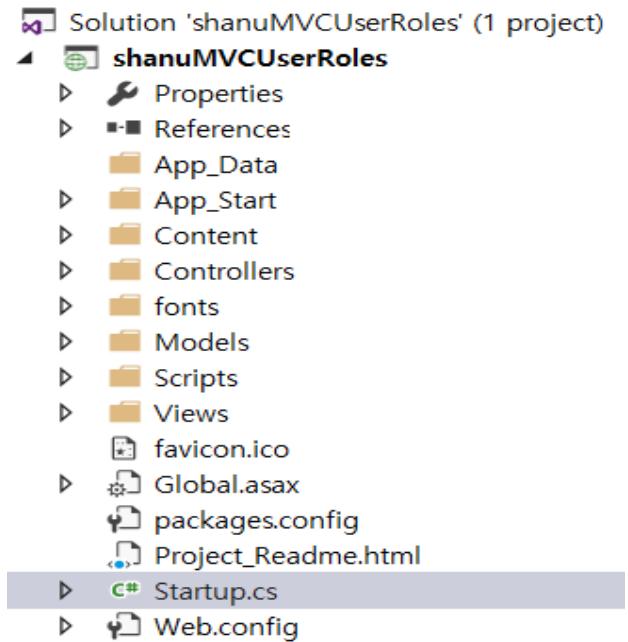
Here in connection string change your SQL Server Name, UID and PWD to create and store all user details in one database.

#### XML

```
<connectionStrings>  
    <add name="DefaultConnection" connectionString="data source=YOURSERVERN  
AME;initial catalog=AttendanceDB;user id=UID;password=PWD;Integrated Securi  
ty=True" providerName="System.Data.SqlClient" />  
</connectionStrings>
```

## Create default Role and Admin User

Firstly, create default user role like "Admin", "Manager", etc and also we will create a default admin user. We will be creating all default roles and user in "Startup.cs"



OWIN (OPEN WEB Interface for .NET) defines a standard interface between .NET and WEB Server and each OWIN application has a **Startup Class** where we can specify components.

### Reference

- [OWIN and Katana](#)

In "Startup.cs" file we can find the Configuration method. From this method we will be calling **ourcreateRolesandUsers()** method to create a default user role and user. We will check for Roles already created or not. If Roles, like Admin, is not created, then we will create a new Role as "Admin" and we will create a default user and set the user role as Admin. We will be using this user as super user where the user can create new roles from our MVC application.

C#

```
public void Configuration(IAppBuilder app)
{
    ConfigureAuth(app);
    createRolesandUsers();
}

// In this method we will create default User roles and Admin user
for login
private void createRolesandUsers()
{
    ApplicationDbContext context = new ApplicationDbContext();

    var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(context));
    var UserManager = new UserManager<ApplicationUser>(new UserStore< ApplicationUser>(context));
```

```

// In Startup iam creating first Admin Role and creating a default Admin User
if (!roleManager.RoleExists("Admin"))
{
    // first we create Admin root
    var role = new Microsoft.AspNet.Identity.EntityFramework.IdentityRole();
    role.Name = "Admin";
    roleManager.Create(role);

    //Here we create a Admin super user who will maintain the website

    var user = new ApplicationUser();
    user.UserName = "shantu";
    user.Email = "syedshanumcain@gmail.com";

    string userPWD = "A@Z200711";

    var chkUser = UserManager.Create(user, userPWD);

    //Add default User to Role Admin
    if (chkUser.Succeeded)
    {
        var result1 = UserManager.AddToRole(user.Id, "Admin");
    }
}

// creating Creating Manager role
if (!roleManager.RoleExists("Manager"))
{
    var role = new Microsoft.AspNet.Identity.EntityFramework.IdentityRole();
    role.Name = "Manager";
    roleManager.Create(role);
}

// creating Creating Employee role
if (!roleManager.RoleExists("Employee"))
{
    var role = new Microsoft.AspNet.Identity.EntityFramework.IdentityRole();
    role.Name = "Employee";
    roleManager.Create(role);
}
}

```

When we run our application we can see new default ASP.NET user related tables will be created in our **AttendanceDB** Database. Here we can see in the following image as all ASP.NET user related tables will be automatically created when we run our application and also all our default user roles will be inserted in AspNetRoles table and default admin user will be created in AspNetUsers table.

AttendanceDB

dbo.AspNetRoles

	Id	Name
1	4f8fb843-ceb8-461c-91dd...	Accounts
2	535fe842-0020-43a4-8de2...	Admin
3	c25a0f27-2742-4c9b-b038...	Employee
4	488bf46d-edb3-4ba6-8c25...	Manager

dbo.AspNetUsers

	Id	Email	EmailConfirmed	PasswordHash	SecurityStamp
1	35cadfb6-18c0-4977-ae37-8ac5a99aa28f	Afraz@123.com	0	AIn1XCZ7W0yMMHW9UNNGewV+zlUH77OL7X9s8B5s...	b99492e2-46b5-439
2	82942037-f0e-4888-a186-9d6730b9e9eb	syedshanumain@gmail.com	0	ABPghRV2gmDf/3y8NCH6SQmxM23l6xg1tlyhSFpzn...	1617a814-12e3-43d

## Customize User Registration with adding username and Role

Register - My ASP.NET Application

localhost:5580/Account/Register

Application name Home About Contact Register Log in

Register.

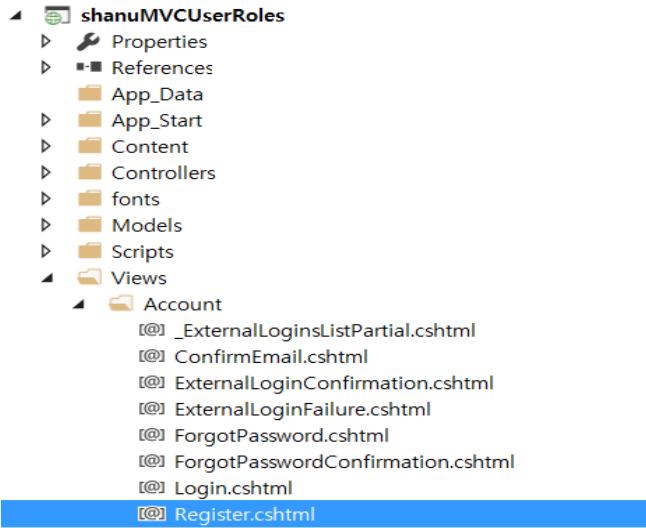
Create a new account.

Email	Afraz@123.com
UserName	Afraz
Password	*****
Confirm password	*****
user Role	Manager

© 2016 - My ASP.NET Application

By default for user registration in ASP.NET MVC 5 we can use Email and password. Here, we will customize the default user registration with adding a username and a ComboBox to display the user roles. User can enter their username and select there user role during registration.

**View Part:** Firstly, add a TextBox for username and ComboBox for displaying User Role in Register.cshtml,



Double click the **Register.cshtml** and change the html code like the following to add textbox and combobox with caption. Here we can see first we add a textbox and Combobox .We bind the combobox with (SelectList) ViewBag.Name.

## HTML

```
@model shanuMVCUserRoles.Models.RegisterViewModel
@{
    ViewBag.Title = "Register";
}

<h2>@ViewBag.Title.</h2>

@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizontal", role = "form" }))
{
    @Html.AntiForgeryToken()
    <h4>Create a new account.</h4>
    <hr />
    @Html.ValidationSummary("", new { @class = "text-danger" })
    <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>

    <div class="form-group">
        @Html.LabelFor(m => m.UserName, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.UserName, new { @class = "form-control" })
        </div>
    </div>
}
```

```
<div class="form-group">
    @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-
label" })
    <div class="col-md-10">
        @Html.PasswordFor(m => m.Password, new { @class = "form-
control" })
    </div>
</div>
<div class="form-group">
    @Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-
2 control-label" })
    <div class="col-md-10">
        @Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-
control" })
    </div>
</div>

<div class="form-group">
    @Html.Label("user Role", new { @class = "col-md-2 control-
label" })
    <div class="col-md-10">
        @* @Html.DropDownList("Name") *@
        @Html.DropDownList("UserRoles", (SelectList)ViewBag.Name, " ")
    </div>
</div>

<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <input type="submit" class="btn btn-
default" value="Register" />
    </div>
</div>
}

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

### Model Part

Next in **AccountViewModel.cs** check for the RegisterViewModel and add the UserRoles and UserName properties with required for validation.

```
3 references
public class RegisterViewModel
{
    [Required]
    [Display(Name = "UserRoles")]
    1 reference
    public string UserRoles { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    3 references
    public string Email { get; set; }

    [Required]
    [Display(Name = "UserName")]
    3 references
    public string UserName { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    3 references
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match")]
}
```

Double click the **AccountViewModel.cs** file from Models folder, find the RegisterViewModel class, add UserName and UserRoles properties as in the following.

C#

```
public class RegisterViewModel
{
    [Required]
    [Display(Name = "UserRoles")]
    public string UserRoles { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }

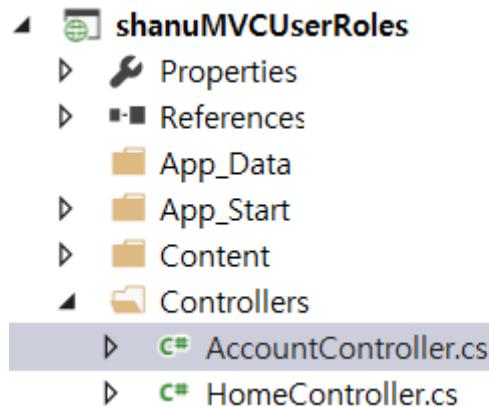
    [Required]
    [Display(Name = "UserName")]
    public string UserName { get; set; }
```

```
[Required]
[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
[DataType(DataType.Password)]
[Display(Name = "Password")]
public string Password { get; set; }

[DataType(DataType.Password)]
[Display(Name = "Confirm password")]
[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
public string ConfirmPassword { get; set; }
}
```

### Controller Part

Next in **AccountController.cs** first we get all the role names to be bound in ComboBox except Admin role and in register button click we will add the functionality to insert username and set user selected role in ASP.NET identity database.



Firstly, create an object for our ApplicationDBContext. Here, ApplicationDBContext is a class which is used to perform all ASP.NET Identity database functions like create user, roles, etc.

C#

```
ApplicationDbContext context;
public AccountController()
{
    context = new ApplicationDbContext();
```

### Register ActionResult method:

Using the **ApplicationDBContext** object we will get all the roles from database. For user registration we will not display the Admin roles. User can select rest of any role type during registration.

C#

```
// GET: /Account/Register
[AllowAnonymous]
public ActionResult Register()
{
    ViewBag.Name = new SelectList(context.Roles.Where(u => !u.Name.
Contains("Admin")))
        .ToList(), "Name", "Name");
    return View();
}
```

## Register User

By default the user email will be stored as username in AspNetUsers table. Here we will change to store the user entered name. After user was created successfully we will set the user selected role for the user.

C#

```
// POST: /Account/Register
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)

{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.UserName,
Email = model.Email };
        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            await SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);

            // For more information on how to enable account confirmation and password reset please visit http://go.microsoft.com/fwlink/?LinkId=320771
            // Send an email with this link
            // string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);
            // var callbackUrl = Url.Action("ConfirmEmail", "Account",
new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

            // await UserManager.SendEmailAsync(user.Id, "Confirm your account", "Please confirm your account by clicking <a href=\"" + callbackUrl + "\">here</a>");
            //Assign Role to user Here
            await this.UserManager.AddToRoleAsync(user.Id, model.UserRoles);
            //Ends Here
            return RedirectToAction("Index", "Users");
        }
        ViewBag.Name = new SelectList(context.Roles.Where(u => !u.Name.
Contains("Admin")))
    }
}
```

```
        .ToList(), "Name", "Name");
        AddErrors(result);
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}
```

## Customize User login

In the same way as user registration we will customize user login to change email as username to enter. By default in ASP.NET MVC 5 for login user needs to enter Email and password. Here we will customize for user by entering username and password. In this demo we are not using any other Facebook, Gmail or Twitter login so we will be using UserName instead of Email.

### View Part

Here we will change the email with UserName in **Login.cshtml**. We can find the **Login.cshtml** file from the folder inside **Views/Account/Login.cshtml**.

C#

```
@using shanuMVCUserRoles.Models
@model LoginViewModel
 @{
    ViewBag.Title = "Log in";
}

<h2>@ViewBag.Title</h2>
<div class="row">
    <div class="col-md-8">
        <section id="loginForm">
            @using (Html.BeginForm("Login", "Account", new { ReturnUrl = ViewBag.ReturnUrl }, FormMethod.Post, new { @class = "form-horizontal", role = "form" }))
            {
                @Html.AntiForgeryToken()
                <h4>Use a local account to log in.</h4>
                <hr />
                @Html.ValidationSummary(true, "", new { @class = "text-danger" })
                <div class="form-group">
                    @Html.LabelFor(m => m.UserName, new { @class = "col-md-2 control-label" })
                    <div class="col-md-10">
                        @Html.TextBoxFor(m => m.UserName, new { @class = "form-control" })
                        @Html.ValidationMessageFor(m => m.UserName, "", new { @class = "text-danger" })
                    </div>
                </div>
                <div class="form-group">
                    @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
```

```
<div class="col-md-10">
    @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
    @Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })
</div>
<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <div class="checkbox">
            @Html.CheckBoxFor(m => m.RememberMe)
            @Html.LabelFor(m => m.RememberMe)
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" value="Log in" class="btn btn-default" />
        </div>
    </div>
    <p>
        @Html.ActionLink("Register as a new user", "Register")
    </p>
    /* Enable this once you have account confirmation enabled f
    or password reset functionality
    <p>
        @Html.ActionLink("Forgot your password?", "ForgotPa
ssword")
    </p>*@
}
</section>
</div>
<div class="col-md-4">
    <section id="socialLoginForm">
        @Html.Partial("_ExternalLoginsListPartial", new ExternalLoginLi
stViewModel { ReturnUrl = ViewBag.ReturnUrl })
    </section>
</div>
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

### Model Part

Same as Registration in AccountViewModel we need to find the loginViewModel to change the Email with UserName,

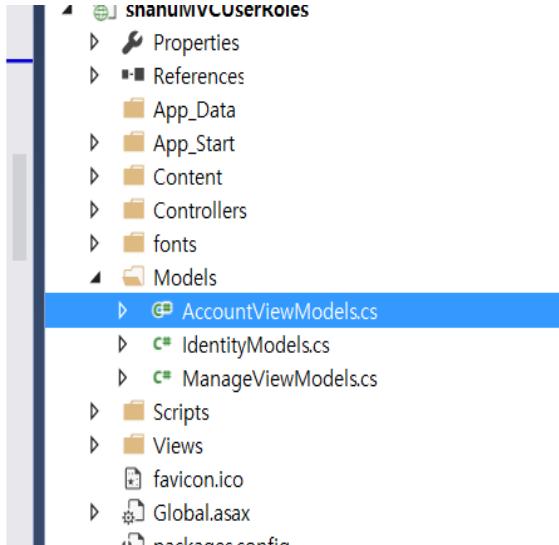
## MVC ASP.NET

```
3 references
public class LoginViewModel
{
    [Required]
    [Display(Name = "UserName")]

    4 references
    public string UserName { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    4 references
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    4 references
    public bool RememberMe { get; set; }
}
```



Here in the following code we can see that we have changed the Email property to UserName.

C#

```
public class LoginViewModel
{
    [Required]
    [Display(Name = "UserName")]

    public string UserName { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    public bool RememberMe { get; set; }
}
```

### Controller Part:

In login button click we need to change the email with username to check from database for user Authentication. Here in the following code we can see as we changed the email with username after successful login we will be redirect to the user page. Next we will see how to create a user page and display the text and menu by user role.

C#

```
// POST: /Account/Login
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
```

```
public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    // This doesn't count login failures towards account lockout
    // To enable password failures to trigger account lockout, change to shouldLockout: true
    var result = await SignInManager.PasswordSignInAsync(model.UserName, model.Password, model.RememberMe, shouldLockout: false);
    switch (result)
    {
        case SignInStatus.Success:
            return RedirectToLocal(returnUrl);
        case SignInStatus.LockedOut:
            return View("Lockout");
        case SignInStatus.RequiresVerification:
            return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });
        case SignInStatus.Failure:
        default:
            ModelState.AddModelError("", "Invalid login attempt.");
    }

    return View(model);
}

// 
// GET: /Account/VerifyCode
[AllowAnonymous]
public async Task<ActionResult> VerifyCode(string provider, string returnUrl, bool rememberMe)
{
    // Require that the user has already logged in via username/password or external login
    if (!await SignInManager.Has Been VerifiedAsync())
    {
        return View("Error");
    }
    return View(new VerifyCodeViewModel { Provider = provider, ReturnUrl = returnUrl, RememberMe = rememberMe });
}
```

## Authenticated and Authorized User page

Here we create a new page for displaying message of Authenticated and Authorized user by their role.

If the logged in user role is Admin, then we will display the welcome message for Admin and display the menu for creating new roles.

If the logged in users roles are Manager, Employee, Accounts, etc. then we will display a welcome message for them.

Firstly, create a new Empty Controller named "**userscontroller.cs**". In this controller first we add the [Authorize] at the top of controller for checking the valid users.

Creating our View: Right click on index ActionResult and create a view .

In view we check for the ViewBag.displayMenu value. If the value is "**Yes**", then we display the Admin welcome message and a link for creating new Menu. If the **ViewBag.displayMenu** is "**No**", then display other users name with welcome message.

### HTML

```
@{  
    ViewBag.Title = "Index";  
}  
  
@if (ViewBag.displayMenu == "Yes")  
{  
    <h1>Welcome Admin. Now you can create user Role.</h1>  
    <h3>  
        <li>@Html.ActionLink("Manage Role", "Index", "Role")</li>  
    </h3>  
}  
else  
{  
    <h2> Welcome <strong>@ViewBag.Name</strong> :) .We will add user module soon </h2>  
}
```

### Controller part

In controller we will check the user is logged in to the system or not. If the user did not log in, then

Display the message as "**Not Logged In**" and if the user is authenticated, then we check the logged in users role. If the users role is "Admin", then we set **ViewBag.displayMenu = "Yes"**, else we set **ViewBag.displayMenu = "No"**.

### C#

```
public ActionResult Index()  
{  
    if (User.Identity.IsAuthenticated)  
    {  
        var user = User.Identity;  
        ViewBag.Name = user.Name;  
  
        ViewBag.displayMenu = "No";  
  
        if (isAdminUser())  
        {  
            ViewBag.displayMenu = "Yes";  
        }  
    }  
}
```

```
        }
        return View();
    }
else
{
    ViewBag.Name = "Not Logged IN";
}
return View();

}
```

For checking the user is logged in we create method and return the Boolean value to our main Index method.

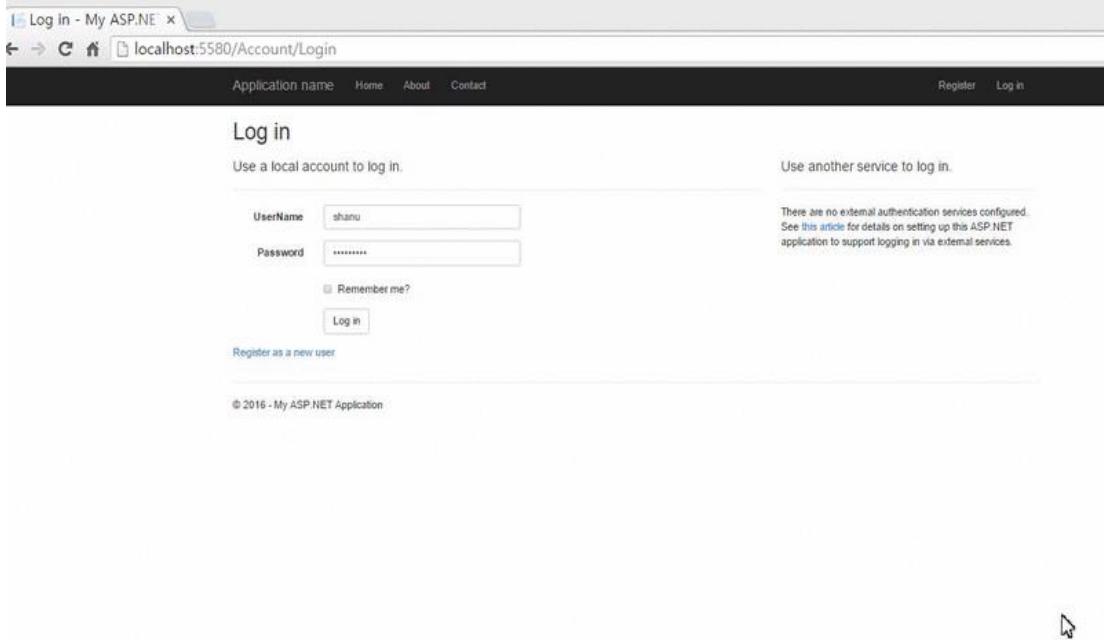
C#

```
public Boolean isAdminUser()
{
    if (User.Identity.IsAuthenticated)
    {
        var user = User.Identity;
        ApplicationDbContext context = new ApplicationDbContext();

        var UserManager = new UserManager<ApplicationUser>(new User
Store<ApplicationUser>(context));
        var s = UserManager.GetRoles(user.GetUserId());
        if (s[0].ToString() == "Admin")
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    return false;
}
```

## Admin users can create Roles

We already saw that if the Admin user is logged in then we will display the link for creating new users. For admin login we have already created a default user with UserName as "**shantu**" and password as "**A@Z200711**",



For creating user role by admin first we will add a new empty controller and named it RoleController.cs,

In this controller we check that the user role is Admin. If the logged in user role is Admin, then we will get all the role names using ApplicationDbContext object.

C#

```
public ActionResult Index()
{
    if (User.Identity.IsAuthenticated)
    {
        if (!isAdminUser())
        {
            return RedirectToAction("Index", "Home");
        }
    }
    else
    {
        return RedirectToAction("Index", "Home");
    }

    var Roles = context.Roles.ToList();
    return View(Roles);
}
```

In view we bind all the user roles inside html table.

HTML

```
@model IEnumerable<Microsoft.AspNet.Identity.EntityFramework.IdentityRole>

@{
    ViewBag.Title = "Add Role";
}


|                                                                                                                                                                                                                                 |            |                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|---------------------------------------------------|
| <h2>Create User Roles</h2>                                                                                                                                                                                                      |            |                                                   |
| <table id="tbrole" style="width:100%; border:dotted 1px; background-color:gainsboro; padding-left:10px;"> @foreach (var item in Model) {     <tr> <td style="width:100%; border:dotted 1px;"> @item.Name </td> </tr> } </table> | @item.Name | <h3><a href="#">Click to Create New Role</a></h3> |
| @item.Name                                                                                                                                                                                                                      |            |                                                   |


```