ASP.NET MVC Tutoriels

tutoriels ASP.NET MVC couvrent toutes les fonctionnalités de ASP.NET MVC. Vous apprendrez de base pour faire avancer les caractéristiques de niveau de ASP.Net MVC. tutoriels de base ont utilisé MVC 5, mais il est applicable à toutes les versions précédentes et les prochaines versions de MVC ainsi.

Pour qui?

Ces tutoriels sont conçus pour les débutants et les professionnels qui veulent apprendre ASP.NET MVC 5 étape par étape.

Conditions préalables:

Connaissance de base de .Net Framework3.5 / 4.5, C #, Visual Studio est nécessaire.

Quiz

Testez vos connaissances ASP.NET MVC 5 - [Lancer MVC test](http://www.tutorialsteacher.com/online-test/mvc-test)

Cliquez sur Suivant pour commencer.

[MVC Architecture](http://www.tutorialsteacher.com/mvc/mvc-architecture):

Dans cette section, vous aurez une vue d'ensemble de l'[architecture MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture). Le modèle architectural[MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture) existe depuis longtemps dans le génie logiciel. Tous les plus toutes les langues utilisent [MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture)avec une légère variation, mais conceptuellement, il reste le même.

Essayons de comprendre l'[architecture MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture) dans ASP.NET.

[MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture) signifie Modèle, Vue et Contrôleur. [MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture) sépare l'application en trois composantes - Modèle, Vue et Contrôleur.

**Modèle** : Modèle représente la forme de la logique des données et des affaires. Elle maintient les données de l'application. Objets de modèle récupérer et de l’état de modèle de magasin dans une base de données.

Le modèle est une logique de données et d'affaires.

**Vue** : View est une interface utilisateur. Afficher les données d'affichage en utilisant le modèle à l'utilisateur et leur permet également de modifier les données.

View est une interface utilisateur.

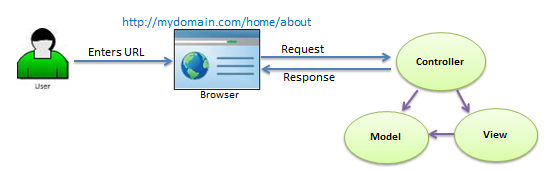
**Contrôleur** : contrôleur gère la demande de l’utilisateur. En règle générale, l’utilisateur d’interagir avec View, qui pose en sterne demande d'URL appropriée, cette demande sera traitée par un contrôleur. Le contrôleur rend la vue appropriée avec les données du modèle en tant que réponse.

Le contrôleur est un gestionnaire de requêtes.

La figure suivante illustre l'interaction entre Modèle, Vue et Contrôleur.

[](http://www.tutorialsteacher.com/Content/images/mvc/mvc-architecture.png)[architecture MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture)

La figure suivante illustre le flux de la demande de l'utilisateur dans [ASP.NET MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture).

[](http://www.tutorialsteacher.com/Content/images/mvc/request-handling-in-mvc.png)Demande / réponse en [architecture MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture)

Conformément à la figure ci-dessus, lorsque l'utilisateur entre une URL dans le navigateur, il va vers le serveur et appelle contrôleur approprié. Ensuite, le contrôleur utilise la vue et le modèle approprié et crée la réponse et l'envoie à l'utilisateur. Nous allons voir les détails de l'interaction dans les prochaines sections.



Points à retenir:

1. [MVC](http://www.tutorialsteacher.com/mvc/mvc-architecture) signifie Modèle, Vue et Contrôleur.
2. Modèle est responsable du maintien des données d'application et la logique métier.
3. La vue est une interface utilisateur de l'application, qui affiche les données.
4. Controller gère les demandes des utilisateurs et rend View appropriée avec des données de modèle.

[ASP.NET MVC Version Histoire](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) :

Microsoft avait introduit [ASP.NET MVC](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) en .Net 3.5, depuis lors , beaucoup de nouvelles fonctionnalités ont été ajoutées.

La liste de table brève histoire suivante de [ASP.NET MVC](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) .

| [**MVC**](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history)**Version** | **Visual Studio** | **.Net Version** | **Date de sortie** | **Caractéristiques** |
| --- | --- | --- | --- | --- |
| [MVC](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) 1.0 | VS2008 | .Net 3.5 | 13-Mar-2009 | * [MVC](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) architecture à moteur webform * routage * HTML Helpers * ajax Helpers * liaison automatique |
| [MVC](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) 2.0 | VS 2008, | .Net 3.5 / 4.0 | 10-Mar-2010 | * Région * contrôleur Asynchronous * méthodes d'assistance Html avec expression lambda * DataAnnotations attributs * validation côté client * Modèle personnalisé * Échafaudage |
| [MVC](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) 3.0 | VS 2010 | .Net 4.0 | 13-Jan-2011 | * validation javascript Unobtrusive * Razor moteur de vue * Les filtres globaux * validation à distance * Dépendance résolveur pour IoC * ViewBag |
| [MVC](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) 4.0 | VS 2010 SP1,  VS 2012 | .NET 4.0 / 4.5 | 15 août 2012 | * modèle de projet mobile * Bundling et minification * Support pour Windows Azure SDK |
| [MVC](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) 5.0 | VS 2013 | .NET 4.5 | 17-oct-2013 | * filtres d'authentification * soutien Bootstrap * Nouveaux objets d'échafaudage * ASP.Net Identity |
| [**MVC**](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history)**5.2** - Courant | VS 2013 | .NET 4.5 | 28-Aug-2014 | * Attribut routage basé * corrections de bugs et fonctionnalités mineures Upate |

Créons d'abord simples [MVC](http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history) application dans la section suivante.

# Créer première application [ASP.NET MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application):

Dans cette section, nous allons créer une nouvelle application [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) 5 avec Visual Studio 2013 pour Web et de comprendre les éléments de base d'une application [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application).

Tout d'abord, la configuration d'un environnement de développement pour développer une application[ASP.NET MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) 5.

**Remarque:**Des tutoriels utilisent [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) 5.2. Cependant, tous les tutoriels de base sont applicables à la version précédente également, comme [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) 3.0 & [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) 4.0.

## **Environnement de développement configuration:**

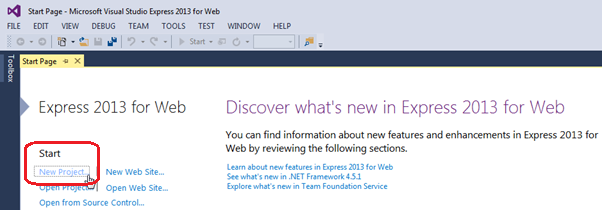
Vous pouvez développer des applications [ASP.NET MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) avec la version appropriée de Visual Studio et framework .NET, comme vous l'avez vu dans la section précédente de l'histoire de version.

Ici, nous allons utiliser [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) v5.2, Visual Studio 2013 pour l'édition et .NET Web Express Framework 4.5 pour créer l'application [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application).

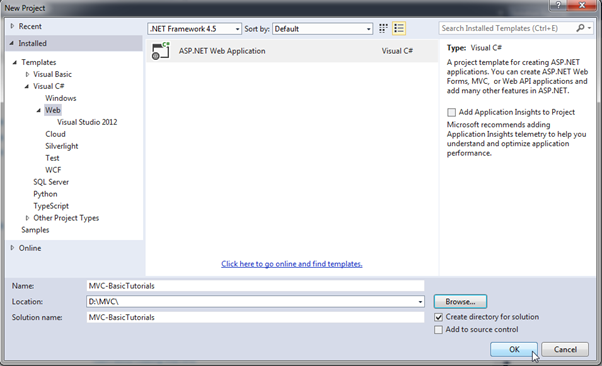
[Télécharger et installer Visual Studio Express 2013](http://www.visualstudio.com/downloads/download-visual-studio-vs) pour le Web-Update3. S'il vous plaît vérifier les[exigences du système](http://www.visualstudio.com/en-us/products/visual-studio-express-vs) pour installer VS2013.

## **Créer la première application**[**MVC**](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application)**simple:**

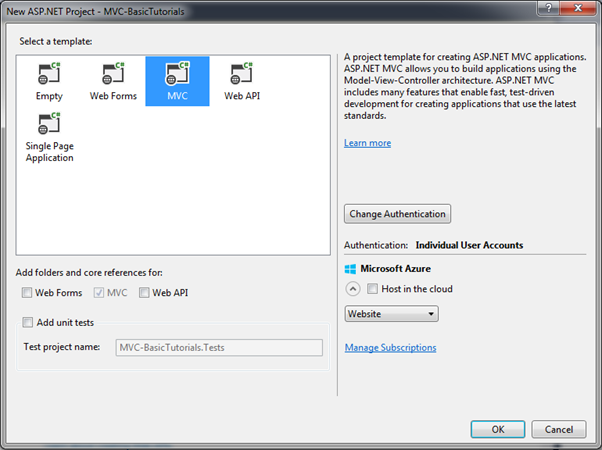
Tout d' abord, ouvrez un Visual Studio 2013 pour Web et cliquez sur un **nouveau projet** lien dans la**Démarrer** page. Alternativement, vous pouvez également sélectionner le **menu Fichier** ->**Nouveau projet** .

[](http://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app1.png)Première application [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application)

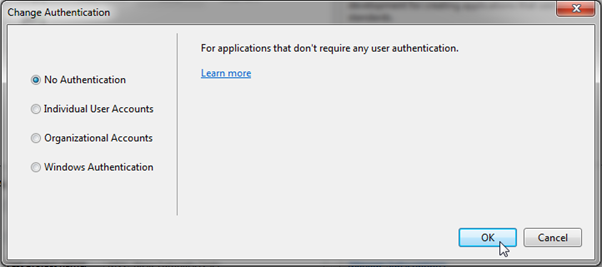
Dans le **nouveau projet** de dialogue comme illustré ci - dessous, développez C # noeud visuel et sélectionnez **Web** dans le volet gauche, puis sélectionnez **Application Web ASP.NET** dans le volet central. Nom de votre projet [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application)-BasicTutorials. (Vous pouvez donner un nom approprié pour votre application). En outre, vous pouvez modifier l'emplacement de l'application [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) en cliquant sur**Parcourir ..** bouton. Enfin, cliquez sur **OK.**

[](http://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app2.png)Sélectionnez le modèle

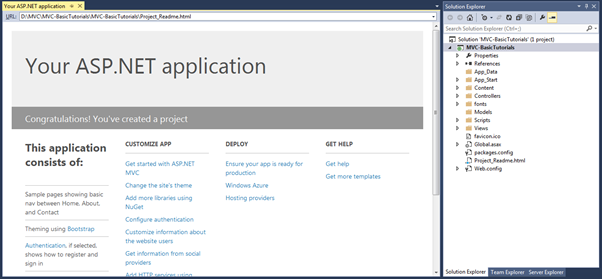
De la **Nouvelle ASP.NET projet** de dialogue, sélectionnez [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) comme indiqué ci - dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app3.png)Sélectionnez [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) modèle

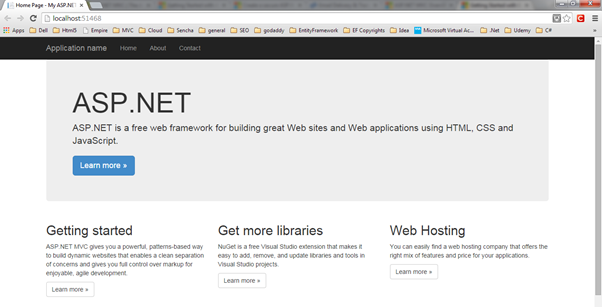
Vous pouvez également modifier l'authentification en cliquant sur l' **authentification Change**bouton. Vous pouvez sélectionner le mode d'authentification approprié pour votre application. Ici, on ne va pas avoir d'authentification pour notre application. Donc , sélectionnez **Pas d'authentification** bouton radio et cliquez sur **OK** . (Par défaut, les comptes d'utilisateurs individuels seront sélectionnés pour authentifier les utilisateurs stockés dans la base de données SQL Server.)

[](http://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app4.png)Sélectionnez Authenctication type

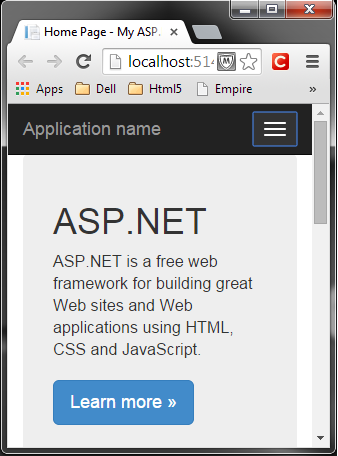
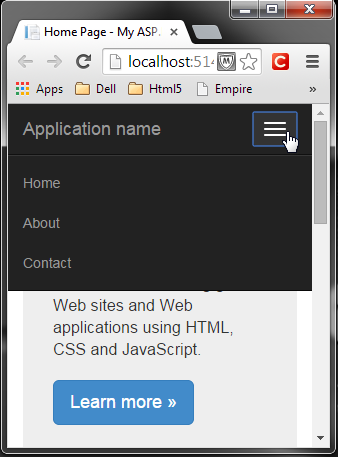
Attendez un certain temps jusqu'à Visual Studio crée un projet [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) simple en utilisant le modèle par défaut, comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app6.png)Première application [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application)

Maintenant, appuyez sur F5 pour exécuter le projet en mode débogage ou Ctrl + F5 pour exécuter le projet sans débogage. Il ouvrira la page d'accueil dans le navigateur comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app7.png)Exécutez [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) application

[MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) 5 projet comprend les fichiers JavaScript et CSS de bootstrap 3.0 par défaut. Ainsi, vous pouvez créer des pages web sensibles. Cette interface utilisateur réactive va changer son look et se sentent en fonction de la taille de l'écran des différents dispositifs. Par exemple, la barre supérieure de menu sera changé dans les appareils mobiles, comme indiqué ci-dessous.

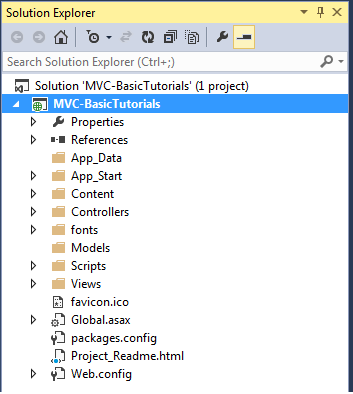
[](http://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app8.png)Respo[](http://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app9.png)nsive [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) applicationResponsive [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) application

Donc, de cette façon, vous pouvez créer votre première application [MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) 5 en utilisant Visual Studio 2013 pour Web.

En savoir plus sur la structure du dossier [ASP.NET MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) dans la section suivante.

# ASP.NET [MVC Folder Structure](http://www.tutorialsteacher.com/mvc/mvc-folder-structure) :

Nous avons créé notre première [MVC](http://www.tutorialsteacher.com/mvc/mvc-folder-structure) demande 5 dans la section précédente. Visual Studio crée la structure de dossier suivant pour [MVC](http://www.tutorialsteacher.com/mvc/mvc-folder-structure) l' application par défaut.

[](http://www.tutorialsteacher.com/Content/images/mvc/mvc-folder-structure.png)[Structure MVC Folder](http://www.tutorialsteacher.com/mvc/mvc-folder-structure)

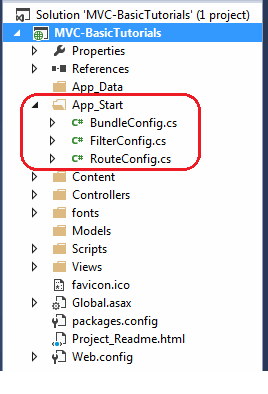
Voyons voir la signification de chaque dossier.

## **Données d'application:**

dossier App\_Data peut contenir des fichiers de données d'application comme LocalDB, les fichiers .mdf, fichiers XML et d'autres fichiers de données connexes. IIS ne sera jamais servir des fichiers à partir du dossier App\_Data.

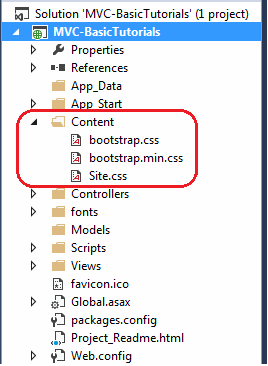
## **App\_Start:**

Dossier App\_Start peut contenir des fichiers de classe qui seront exécutées lorsque l'application démarre. En règle générale, ceux - ci seraient des fichiers de configuration comme AuthConfig.cs, BundleConfig.cs, FilterConfig.cs, RouteConfig.cs etc. [MVC](http://www.tutorialsteacher.com/mvc/mvc-folder-structure) 5 comprend BundleConfig.cs, FilterConfig.cs et RouteConfig.cs par défaut. Nous verrons l' importance de ces fichiers plus tard.

[](http://www.tutorialsteacher.com/Content/images/mvc/appstart.png)App\_Start Folder

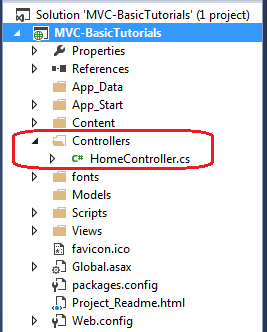
## **Contenu:**

Dossier Contenu contient des fichiers statiques comme les fichiers css, images et fichiers icônes.[MVC](http://www.tutorialsteacher.com/mvc/mvc-folder-structure) demande 5 comprend bootstrap.css, bootstrap.min.css et Site.css par défaut.

[](http://www.tutorialsteacher.com/Content/images/mvc/content-folder.png)Folder Content

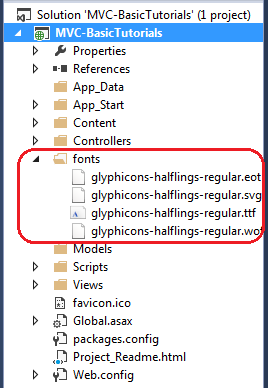
## **Contrôleurs:**

Dossier Controllers contient des fichiers de classe pour les contrôleurs. Contrôleurs poignées de lademande des utilisateurs et renvoie une réponse. [MVC](http://www.tutorialsteacher.com/mvc/mvc-folder-structure) requiert le nom de tous les fichiers du contrôleur pour terminer par "Controller". Vous apprendrez à connaître le contrôleur dans la section suivante.

[](http://www.tutorialsteacher.com/Content/images/mvc/controller-folder.png)dossier Controller

## **polices:**

dossier Polices contient des fichiers de police personnalisé pour votre application.

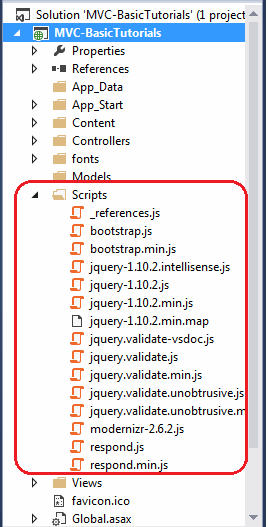
[](http://www.tutorialsteacher.com/content/images/mvc/fonts-folder.png)dossier Polices

## **Des modèles:**

dossier Modèles contient des fichiers de classe modèle. Typiquement classe modèle comprend des propriétés publiques, qui seront utilisés par l'application pour tenir et manipuler des données d'application.

## **Scripts:**

Dossier Scripts contient des fichiers JavaScript ou VBScript pour l'application. [MVC](http://www.tutorialsteacher.com/mvc/mvc-folder-structure) 5 inclut desfichiers javascript pour bootstrap, jquery 1.10 et modernisateur par défaut.

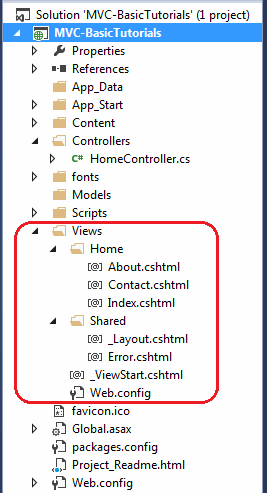
[](http://www.tutorialsteacher.com/Content/images/mvc/scripts-folder.png)Scripts Folder

## **Vues:**

dossier Vues contient des fichiers html pour l'application. Typiquement voir le fichier est un fichier .cshtml où vous écrivez html et code C # ou VB.NET.

dossier Vues comprend dossier séparé pour chaque contrôleur. Par exemple, tous les fichiers .cshtml, qui seront rendus par HomeController seront dans View> dossier Accueil.

dossier partagé sous Affichage dossier contient tous les points de vue qui seront partagés entre les différents contrôleurs, par exemple la mise en page des fichiers.

[](http://www.tutorialsteacher.com/Content/images/mvc/view-folder.png)Voir le dossier

En outre, [MVC](http://www.tutorialsteacher.com/mvc/mvc-folder-structure) projet comprend également des fichiers de configuration suivants:

## **Global.asax:**

Global.asax vous permet d'écrire du code qui fonctionne en réponse aux événements de niveau d'application, tels que Application\_BeginRequest, Application\_Start, Application\_Error, session\_start, Session\_End etc.

## **Packages.config:**

fichier Packages.config est géré par NuGet de garder une trace de ce que les paquets et les versions que vous avez installé dans l'application.

## **Web.config:**

fichier web.config contient des configurations de niveau d'application.

Apprenez comment [MVC](http://www.tutorialsteacher.com/mvc/mvc-folder-structure) cadre gère la demande en utilisant le routage dans la section suivante.

# [Routage dans MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) :

Dans l'application Web Forms ASP.NET, chaque URL doit correspondre à un fichier spécifique .aspx.Par exemple, une URL http: //domain/studentsinfo.aspx doit correspondre avec le fichier studentsinfo.aspx qui contient le code et le balisage pour rendre une réponse au navigateur.

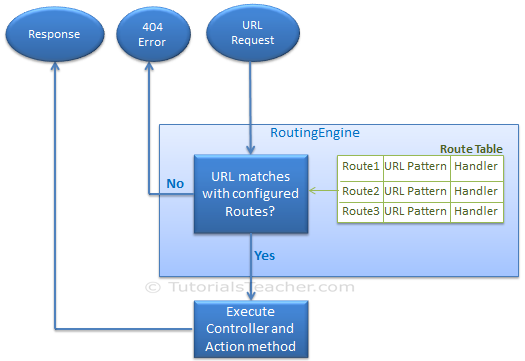
http://www.tutorialsteacher.com/Content/images/tips.png[Routage](http://www.tutorialsteacher.com/mvc/routing-in-mvc) est pas spécifique à[MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) cadre. Il peut être utilisé avec l' application Webform ASP.NET ou [MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) application.

ASP.NET introduit de [routage](http://www.tutorialsteacher.com/mvc/routing-in-mvc) pour éliminer les besoins de mappage de chaque URL avec un fichier physique. [Routage](http://www.tutorialsteacher.com/mvc/routing-in-mvc) nous permettent de définir le format d'URL qui correspond à la gestionnaire de requêtes. Ce gestionnaire de demande peut être un fichier ou une classe. En application Webform ASP.NET, gestionnaire de requêtes est le fichier .aspx et [MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) , il est classe contrôleur et la méthode d' action. Par exemple, http: // domain / étudiants peuvent être mappées à http: //domain/studentsinfo.aspx dans ASP.NET Webforms et la même URL peuvent être mappés Controller étudiant et Index méthode d'action dans [MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) .

## **Route:**

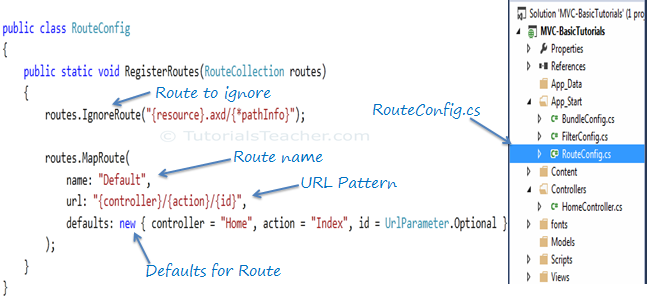
Route définit le modèle d'URL et les informations de gestionnaire. Tous les itinéraires configurés d'une application stockés dans RouteTable et seront utilisés par [Routage](http://www.tutorialsteacher.com/mvc/routing-in-mvc) moteur pour déterminer la classe de gestionnaire approprié ou le fichier pour une requête entrante.

La figure suivante illustre le [routage](http://www.tutorialsteacher.com/mvc/routing-in-mvc) processus.

[](http://www.tutorialsteacher.com/Content/images/mvc/routing-process.png)[Routage dans MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc)

## **Configurer Route:**

Chaque [MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) application doit configurer (registre) au moins une route, qui est configuré par [MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc)cadre par défaut. Vous pouvez enregistrer un itinéraire dans**RouteConfig** classe, qui est en RouteConfig.cs sous **App\_Start** dossier. La figure suivante illustre comment configurer un itinéraire dans la classe RouteConfig.

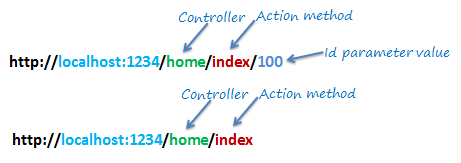
[](http://www.tutorialsteacher.com/Content/images/mvc/routeconfig.png)Configurer Route dans [MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc)

Comme vous pouvez le voir dans la figure ci - dessus, l'itinéraire est configuré à l' aide du MapRoute () méthode d'extension du RouteCollection, où nom est "Default", modèle d'URL est *"{contrôleur} / {action} / {id}"* et par défaut paramètre pour contrôleur, méthode d'action et paramètre id. La valeur par défaut spécifie quel contrôleur, méthode d'action ou de la valeur du paramètre id doivent être utilisées si elles n'existent pas dans la requête URL entrante.

De la même façon, vous pouvez configurer d' autres voies en utilisant la méthode MapRoute de RouteCollection. Ce RouteCollection est en fait une propriété de [RouteTable](http://msdn.microsoft.com/en-us/library/system.web.routing.routetable(v=vs.110).aspx) classe.

### **Motif URL:**

Le modèle d'URL est considéré seulement après partie nom de domaine dans l'URL. Par exemple, le modèle d'URL *"{contrôleur} / {action} / {id}"* ressemblerait localhost: 1234 / {contrôleur} / {action} / {id}. Tout ce qui suit "localhost: 1234 /" serait considéré comme nom du contrôleur. De la même façon, quoi que ce soit après le nom du contrôleur serait considéré comme nom d'action et la valeur du paramètre id.

[](http://www.tutorialsteacher.com/Content/images/mvc/url-routing.png)[Routage dans MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc)

Si l'URL ne contient rien après le nom de domaine, le contrôleur par défaut et la méthode d'action se chargera de la demande. Par exemple, http: // lcoalhost: 1234 serait géré par HomeController et Index méthode tel que configuré dans le paramètre par défaut.

Le tableau ci-après qui Controller, la méthode d'action et le paramètre Id traiteraient différentes URL ci-dessus envisagent route par défaut.

| **URL** | **Manette** | **action** | **ça** |
| --- | --- | --- | --- |
| http: // localhost / home | HomeController | Indice | nul |
| http: // localhost / home / index / 123 | HomeController | Indice | 123 |
| http: // localhost / home / about | HomeController | Sur | nul |
| http: // localhost / home / contact | HomeController | Contact | nul |
| http: // localhost / étudiant | StudentController | Indice | nul |
| http: // localhost / étudiant / modifier / 123 | StudentController | modifier | 123 |

## **Différentes Voies:**

Vous pouvez également configurer un itinéraire personnalisé en utilisant la méthode d'extension MapRoute. Vous devez fournir au moins deux paramètres dans MapRoute, nom de la route et le modèle url. Le paramètre par défaut est facultative.

Vous pouvez enregistrer plusieurs itinéraires personnalisés avec des noms différents. Prenons l'exemple suivant où nous enregistrons route "étudiant".

Itinéraires personnalisés Exemple:

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Student",

url: "students/{id}",

defaults: new { controller = "Student", action = "Index"}

);

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

}

}

Comme le montre le code ci - dessus, le motif d'URL pour l'itinéraire des étudiants est *étudiants / {id}* , qui spécifie que toute URL qui commence par DOMAINNAME / étudiants, doit être manipulé par StudentController. Notez que nous n'avons pas spécifié {action} dans le modèle d'URL parce que nous voulons que chaque URL qui commence par l' étudiant doit toujours utiliser Index action de StudentController. Nous avons spécifié contrôleur et l' action par défaut pour traiter toute demande d'URL qui commence à partir de domainname / étudiants.

[MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) cadre évalue chaque itinéraire en séquence. Il commence avec le premier itinéraire configuré et si url entrante ne satisfait pas le modèle d'URL de la route , alors il évaluera deuxième voie et ainsi de suite. Dans l'exemple ci - dessus, le [routage](http://www.tutorialsteacher.com/mvc/routing-in-mvc) moteur évaluera la route de l' étudiant d' abord et si url entrant ne commence pas par / étudiants alors seulement il examinera la deuxième voie qui est la route par défaut.

Le tableau suivant montre comment les différentes URL sera mappé à la route de l'élève:

| **URL** | **Manette** | **action** | **ça** |
| --- | --- | --- | --- |
| http: // localhost / étudiants / 123 | StudentController | Indice | 123 |
| http: // localhost / étudiants / index / 123 | StudentController | Indice | 123 |
| http: // localhost / étudiants Id = 123? | StudentController | Indice | 123 |

## **Contraintes d'itinéraire:**

Vous pouvez également appliquer des restrictions à la valeur du paramètre en configurant les contraintes d'itinéraire. Par exemple, l'itinéraire suivant applique une restriction sur le paramètre id que la valeur d'un identifiant doit être numérique.

Exemple: Contraintes Route

routes.MapRoute(

name: "Student",

url: "student/{id}/{name}/{standardId}",

defaults: new { controller = "Student", action = "Index", id = UrlParameter.Optional, name = UrlParameter.Optional, standardId = UrlParameter.Optional },

constraints: new { id = @"\d+" }

);

Donc , si vous donnez une valeur non numérique pour le paramètre id alors que la demande sera traitée par une autre voie ou, s'il n'y a pas de voies correspondant alors *"La ressource n'a pu être trouvée"* erreur sera levée.

## **Inscription Routes:**

Maintenant, après avoir configuré toutes les routes en classe RouteConfig, vous devez vous inscrire dans l'événement Application\_Start () dans le Global.asax. Alors qu'elle comprend tous vos itinéraires en RouteTable.

Exemple: Route Inscription

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

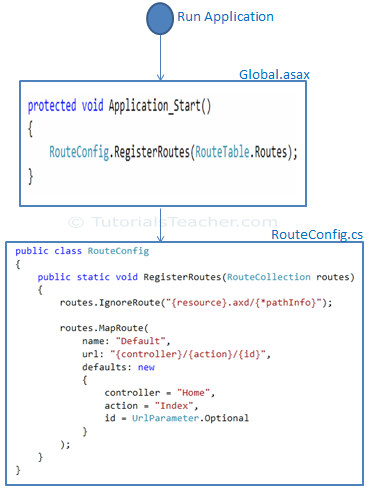
{

RouteConfig.RegisterRoutes(RouteTable.Routes);

}

}

La figure suivante illustre processus d'inscription Route.

[](http://www.tutorialsteacher.com/Content/images/mvc/Route-configuration-process.png)Inscrivez-Route

Ainsi, le [routage](http://www.tutorialsteacher.com/mvc/routing-in-mvc) joue un rôle important dans [MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) cadre.



### **Points à retenir:**

1. [Routing](http://www.tutorialsteacher.com/mvc/routing-in-mvc) joue un rôle important dans [MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) cadre. [Routage](http://www.tutorialsteacher.com/mvc/routing-in-mvc) maps URL vers un fichier physique ou classe (classe de contrôleur dans [MVC](http://www.tutorialsteacher.com/mvc/routing-in-mvc) ).
2. Route contient modèle d'URL et les informations de gestionnaire. motif URL commence après le nom de domaine.
3. Les itinéraires peuvent être configurés en classe RouteConfig. itinéraires personnalisés multiples peuvent également être configurés.
4. les contraintes de Route applique des restrictions sur la valeur des paramètres.
5. Route doit être enregistré dans l'événement Application\_Start dans le fichier Global.ascx.cs.

En savoir plus sur le contrôleur dans la section suivante.

# Manette:

Dans cette section, vous allez apprendre davantage sur le **contrôleur** dans ASP.NET MVC.

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

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 situé dans le dossier Controller de la structure du dossier MVC.

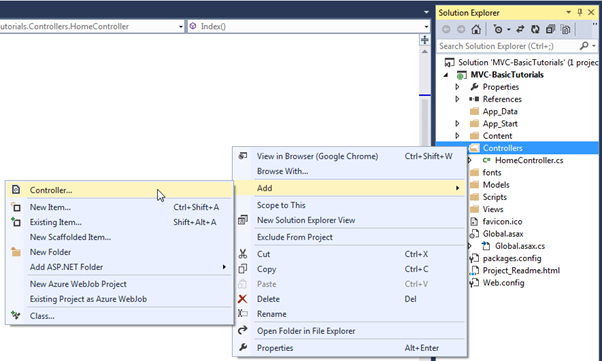
http://www.tutorialsteacher.com/Content/images/tips.pngMVC va jeter "La ressource ne peut pas être trouvé" erreur lorsque vous n'ajoutez pas "Controller" au nom de la classe de contrôleur.

## **Ajout d'un nouveau contrôleur:**

Maintenant, nous allons ajouter un nouveau contrôleur vide à l'aide de Visual Studio 2013 pour Web et MVC 5.

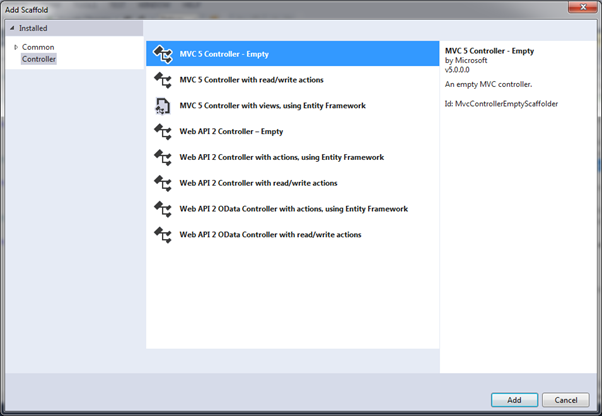
Dans la section précédente, nous avons appris comment créer notre première application MVC, qui à son tour a créé une HomeController par défaut. Ici, nous allons créer un nouveau StudentController.

Dans Visual Studio, cliquez droit sur le **contrôleur** dossier -> sélectionnez **Ajouter** -> cliquez sur**Controller ..**

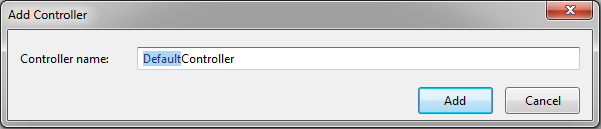
[](http://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-1.png)Ajouter un nouveau contrôleur

Cela ouvre Ajouter dialogue Échafaudages comme illustré ci-dessous. (Visual Studio 2013 a introduit la boîte de dialogue Ajouter un nouvel élément Échafaudages. Ce dialogue a remplacé l'Add View / Add dialogue Contrôleurs vu dans la version antérieure de Visual Studio.)

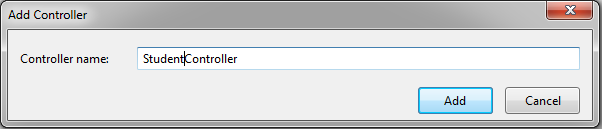
**Remarque:**Echafaudage est un cadre de génération automatique de code pour les applications Web ASP.NET.Echafaudage réduit le temps nécessaire pour développer un contrôleur, voir etc dans framework MVC. Vous pouvez développer un modèle d'échafaudage personnalisé en utilisant des modèles T4 selon votre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

[](http://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-2.png)Manette

Ajouter dialogue Échafaudages contient différents modèles pour créer un nouveau **contrôleur** . Nous allons en apprendre davantage sur d' autres modèles plus tard. Pour l' instant, sélectionnez **"MVC 5 Controller - Empty"** et cliquez sur **Ajouter** . Il ouvrira boîte de dialogue Ajouter Controller comme indiqué ci - dessous

[](http://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-3.png)Manette

Dans la boîte de dialogue Add Controller, entrez le nom du contrôleur. Rappelez - vous, le nom du contrôleur doit se terminer avec le contrôleur. Entrons StudentController et cliquez sur **Ajouter** .

[](http://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-4.png)Manette

Cela va créer la classe StudentController avec la méthode Index dans le fichier StudentController.cs sous dossier Controllers, comme indiqué ci-dessous.

StudentController en C #:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

{

return View();

}

}

}

Comme vous pouvez le voir ci-dessus, la classe StudentController est dérivée de la classe Controller.Chaque contrôleur MVC doit dérivé de cette classe de contrôleur abstrait. Cette classe de contrôleur de base contient des méthodes d'assistance qui peuvent être utilisés à des fins diverses.

Maintenant, nous allons retourner une chaîne fictive de l'index de la méthode ci-dessus StudentController d'action. Modification du type de méthode Index de retour de ActionResult en chaîne et en retournant la chaîne fictive est représentée ci-dessous. Vous apprendrez ActionResult dans la section suivante.

StudentController en C #:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

// GET: Student

public string Index()

{

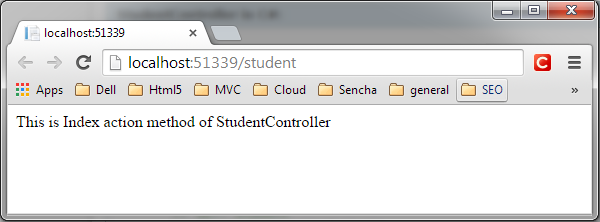
return "This is Index action method of StudentController";

}

}

}

Nous avons déjà vu dans la section de routage que la demande URL *http: // localhost / étudiant* ou*http: // localhost / étudiant / index* est géré par la méthode Index () de la classe StudentController, illustré ci - dessus. Donc , nous allons l' appeler à partir du navigateur et vous verrez la page suivante dans le navigateur.

[](http://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-5.png)Manette



### **Points à retenir:**

1. Un contrôleur gère incomming demandes d'URL. MVC routage envoie une demande au contrôleur approprié et la méthode d'action basé sur l'URL et itinéraires configurés.
2. Toutes les méthodes publiques de la classe Controlle sont appelés **action**méthodes.
3. Une classe de contrôleur doit être dérivé de***System.Web.Mvc.Controller*** classe.
4. Un nom de classe de contrôleur doit se terminer par "Controller".
5. Nouveau contrôleur peut être créé en utilisant différents modèles d'échafaudage. Vous pouvez créer des modèles d'échafaudage personnalisé aussi.

En savoir plus sur **action** méthodes dans la section suivante.

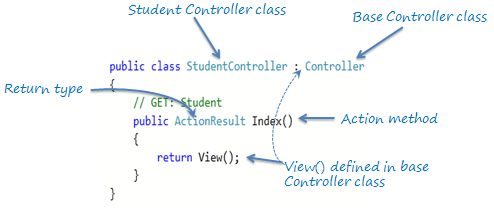
# méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc):

Dans cette section, vous apprendrez au sujet de la méthode de la classe de contrôleur d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc).

Toutes les méthodes publiques d'une classe de contrôleur sont appelées méthodes d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc). Ils sont comme les autres méthodes normales avec les restrictions suivantes:

1. méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) doit être publique. Il ne peut pas être privé ou protégé
2. méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) ne peut pas être surchargé
3. méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) ne peut pas être une méthode statique.

Ce qui suit est un exemple de méthode d'indice StudentController d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc)

[](http://www.tutorialsteacher.com/Content/images/mvc/action-method.png)Méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc)

Comme vous pouvez le voir dans la figure ci-dessus, la méthode Index est une méthode publique et il retourne ActionResult en utilisant la méthode View (). La méthode View () est définie dans la classe de base de contrôleur, qui retourne le ActionResult approprié.

## **Par défaut méthode d'**[**action**](http://www.tutorialsteacher.com/mvc/action-method-in-mvc)**:**

Chaque contrôleur peut avoir une méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) par défaut comme par itinéraire configuré en classe RouteConfig. Par défaut, Index est une méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) par défaut pour un contrôleur, comme par root par défaut configuré comme illustré ci-dessous.

Route par défaut:

routes.MapRoute(

name: "Default",

url: "{controller}/{[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc)}/{id}/{name}",

defaults: new { controller = "Home",

[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) = "Index",

id = UrlParameter.Optional

});

Cependant, vous pouvez modifier le nom de l'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) par défaut selon vos besoins en classe RouteConfig.

## **ActionResult:**

framework [MVC](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) comprend différentes classes de résultat, qui peut être un retour de méthodes d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc). Il 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](http://www.tutorialsteacher.com/mvc/action-method-in-mvc).

| **Résultat classe** | **La description** |
| --- | --- |
| Voir résultat | Représente HTML et le balisage. |
| EmptyResult | Représente Pas de réponse. |
| ContentResult | Représente chaîne littérale. |
| FileContentResult / FilePathResult / FileStreamResult | Représente le contenu d'un fichier |
| JavaScriptResult | Représenter un script JavaScript. |
| JsonResult | Représenter JSON qui peut être utilisé dans Ajax |
| RedirectResult | Représente une redirection vers une nouvelle URL |
| RedirectToRouteResult | Représenter une autre [action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) du même ou un autre contrôleur |
| PartialViewResult | Retourne HTML de la vue partielle |
| HttpUnauthorizedResult | 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](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) qui renvoie tout type de résultat ci-dessus.Cependant, vous pouvez spécifier la classe de résultat approprié comme un type de méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc)de retour.

La méthode Index () de StudentController dans la figure ci-dessus utilise la méthode View () pour revenir ViewResult (qui est dérivé de ActionResult). 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.

| **Résultat classe** | **La description** | **Méthode base contrôle** |
| --- | --- | --- |
| Voir résultat | Représente HTML et le balisage. | Vue() |
| EmptyResult | Représente Pas de réponse. |  |
| ContentResult | Représente chaîne littérale. | Contenu() |
| FileContentResult,  FilePathResult,  FileStreamResult | Représente le contenu d'un fichier | Fichier() |
| JavaScriptResult | Représenter un script JavaScript. | JavaScript () |
| JsonResult | Représenter JSON qui peut être utilisé dans Ajax | Json () |
| RedirectResult | Représente une redirection vers une nouvelle URL | Réorienter() |
| RedirectToRouteResult | Représenter une autre [action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) du même ou un autre contrôleur | RedirectToRoute () |
| PartialViewResult | Renvoie HTML | PartialView () |
| HttpUnauthorizedResult | Retourne HTTP 403 statut |  |

Comme vous pouvez le voir dans le tableau ci-dessus, Voir méthode renvoie ViewResult, méthode Content retourne la chaîne, la méthode de fichier renvoie le contenu d'un fichier et ainsi de suite.Utilisez différentes méthodes mentionnées dans le tableau ci-dessus, afin d'obtenir différents types de résultats à partir d'une méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc).

## **Méthode d'**[**action**](http://www.tutorialsteacher.com/mvc/action-method-in-mvc)**Paramètres:**

Toutes les méthodes d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) 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.

Exemple: méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) paramètres

[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");

}

http://www.tutorialsteacher.com/Content/images/tips.pngparamètres de la méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc)peuvent être de type Nullable.

Par défaut, les valeurs des paramètres de la méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc)sont récupérés à partir de la collecte de données de la demande.La collecte de données comprend le nom / valeurs paires pour lesdonnées de formulaire ou des valeurs de chaîne de requête ou valeurs de cookie. Obligatoire dans[ASP.NET MVC](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) Modèle mappe automatiquement la collecte des données URL de chaîne de requête ou formulaire pour les paramètres de la méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) si les deux noms sélectionnés. Visitez la[liaison modèle](http://www.tutorialsteacher.com/mvc/model-binding-in-asp.net-mvc) section pour plus d' informations à ce sujet .



### **Points à retenir:**

1. Toutes les méthodes publiques de la classe de contrôleur sont appelées méthodes d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc).
2. Méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) a la suite des restrictions.   
       - Méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) doit être publique. Il ne peut pas être privé ou protégé.   
       - Méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) ne peut pas être surchargé.   
       - Méthode d' [action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) ne peut pas être une méthode statique.
3. ActionResult est une classe de base de tout le type de résultat qui revient de la méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc).
4. classe Controller base contient des méthodes qui renvoie type de résultat approprié, par exemple View (), Contenu (), File (), JavaScript (), etc.
5. Méthode d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) peut comprendre [nullables](http://www.tutorialsteacher.com/csharp/csharp-nullable-types) paramètres de type.

En savoir plus sur les sélecteurs d'[action](http://www.tutorialsteacher.com/mvc/action-method-in-mvc) dans la section suivante.

# [Sélecteurs d' action](http://www.tutorialsteacher.com/mvc/action-selectores-in-mvc) :

Le sélecteur d' action est l'attribut qui peut être appliquée aux procédés d'action. Il aide moteur deroutage pour sélectionner la méthode d'action correcte de traiter une demande particulière. [MVC](http://www.tutorialsteacher.com/mvc/action-selectores-in-mvc) 5 comprend le sélecteur d'action attributs suivants:

1. ActionName
2. non-action
3. Verbes d'action

## **ActionName:**

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

Exemple: ActionName

public class StudentController : Controller

{

public StudentController()

{

}

[ActionName("find")]

public ActionResult GetById(int id)

{

// get student from the database

return View();

}

}

Dans l'exemple ci - dessus, nous avons appliqué l' ActioName("find")attribut à la méthode d'action GetById. Alors maintenant, le nom de l' action est "trouver" au lieu de "GetById". Cette méthode d'action sera invoquée sur *http: // localhost / étudiant / find / 1* demande au lieu de *http: // localhost / étudiant / getbyid / 1* demande.

## **non-action:**

attribut de sélection de non-action 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.

Par exemple, le GetStudent () méthode public ne peut être invoqué de la même manière que la méthode d'action dans l'exemple suivant.

Exemple: non-action

public class StudentController : Controller

{

public StudentController()

{

}

[NonAction]

public Student GetStudnet(int id)

{

return studentList.Where(s => s.StudentId == id).FirstOrDefault();

}

}



### **Points à retenir:**

1. [MVC](http://www.tutorialsteacher.com/mvc/action-selectores-in-mvc) moteur framework de routage utilise [action sélecteurs](http://www.tutorialsteacher.com/mvc/action-selectores-in-mvc) attributs pour déterminer la méthode action pour invoquer.
2. Trois [sélecteurs d'action](http://www.tutorialsteacher.com/mvc/action-selectores-in-mvc) attributs sont disponibles dans [MVC](http://www.tutorialsteacher.com/mvc/action-selectores-in-mvc) 5  
      - actionName   
      - non - action   
      - ActionVerbs
3. ActionName attribut est utilisé pour spécifier un nom différent de l'action que le nom de la méthode.
4. attribut non-action marque la méthode publique de la classe de contrôleur comme méthode non-action. Il ne peut pas être invoquée.

En savoir plus sur ActionVerbs sélecteur dans la section suivante.

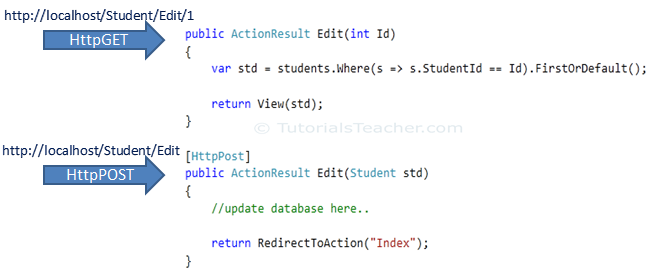
# Verbes d'action:

Dans cette section, vous apprendrez tout sur les [ActionVerbs](http://www.tutorialsteacher.com/mvc/actionverbs-in-mvc) sélecteurs attribuent.

Le sélecteur de [ActionVerbs](http://www.tutorialsteacher.com/mvc/actionverbs-in-mvc) 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](http://www.tutorialsteacher.com/mvc/actionverbs-in-mvc) supporte différents [ActionVerbs](http://www.tutorialsteacher.com/mvc/actionverbs-in-mvc), comme HttpGet, HttpPost, HttpPut, HttpDelete, HttpOptions & 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.

La figure suivante illustre les verbes HttpGet et HttpPost action.

[](http://www.tutorialsteacher.com/Content/images/mvc/actionverbs.png)Verbes d'action

Le tableau suivant présente l'utilisation de méthodes de http:

| **méthode Http** | **Usage** |
| --- | --- |
| OBTENEZ | Pour récupérer les informations à partir du serveur. Les paramètres seront ajoutés dans la chaîne de requête. |
| POSTER | Pour créer une nouvelle ressource. |
| METTRE | Pour mettre à jour une ressource existante. |
| TÊTE | 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. |
| EFFACER | Pour supprimer une ressource existante. |
| PIÈCE | Pour complète ou mise à jour partielle de la ressource. |

Visitez [W3.org](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html) pour plus d' informations sur les méthodes Http.

L'exemple suivant montre les différentes méthodes d'action prend en charge différents [ActionVerbs](http://www.tutorialsteacher.com/mvc/actionverbs-in-mvc):

Exemple: [ActionVerbs](http://www.tutorialsteacher.com/mvc/actionverbs-in-mvc)

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 [ActionVerbs](http://www.tutorialsteacher.com/mvc/actionverbs-in-mvc) dans l'exemple suivant:

Exemple: AcceptVerbs

[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Get)]

public ActionResult GetAndPostAction()

{

return RedirectToAction("Index");

}



### **Points à retenir:**

1. [ActionVerbs](http://www.tutorialsteacher.com/mvc/actionverbs-in-mvc) sont un autre sélecteurs d'action qui sélectionne une méthode d'action basée sur demande par exemple les méthodes POST, GET, PUT etc.
2. méthodes d'action multiples peuvent avoir le même nom avec différents verbes d'action. Méthode des règles de surcharge sont applicables.
3. verbes d'action multiples peuvent être appliquées à une méthode d'action unique en utilisant AcceptVerbs attribut.

En savoir plus sur le modèle dans la section suivante.

# Modèle dans ASP.NET MVC:

Dans cette section, vous apprendrez tout sur le modèle dans le framework ASP.NET MVC.

Modèle représente les données spécifiques de domaine et la logique métier dans l'architecture MVC.Elle maintient les données de l'application. objets de modèle récupérer et de l'état de modèle de magasin dans le magasin de persistence comme une base de données.

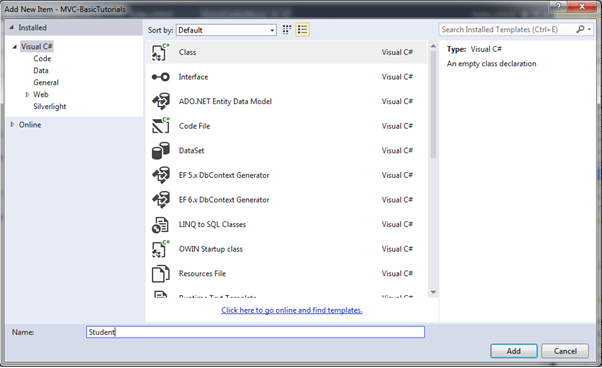
classe de modèle contient des données dans les propriétés publiques. Toutes les classes de modèles résident dans le dossier du modèle dans la structure du dossier MVC.

Voyons voir comment ajouter la classe de modèle dans ASP.NET MVC.

## **L'ajout du modèle:**

Ouvrez notre premier projet MVC créé à l'étape précédente dans Visual Studio. Faites un clic droit sur le dossier Modèle -> Ajouter -> cliquez sur Class ..

Dans la boîte de dialogue Nouveau Ajouter un élément, entrez le nom de classe 'Student' et cliquez sur **Ajouter** .

[](http://www.tutorialsteacher.com/Content/images/mvc/create-model.png)Créer une classe modèle

Cela va ajouter une nouvelle classe d'étudiants dans le dossier du modèle. Maintenant, ajoutez Id, Nom, propriétés de l'âge comme indiqué ci-dessous.

Exemple: classe Model

namespace MVC\_BasicTutorials.Models

{

public class Student

{

public int StudentId { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

}

Donc, de cette façon, vous pouvez créer une classe de modèle que vous pouvez utiliser dans la vue.Vous apprendrez à mettre en œuvre à l'aide de validations modèle plus tard.

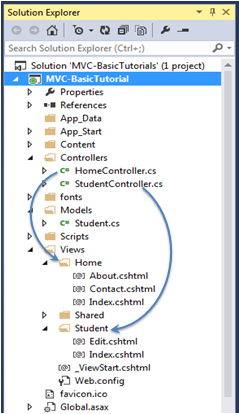
Apprenez à créer une vue dans la section suivante.

# Voir dans ASP.NET MVC:

Dans cette section, vous apprendrez au sujet de la Voir dans le framework ASP.NET MVC.

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 **Vues** dossier. 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, les vues qui seront rendus à partir de StudentController, seront réside dans Vues> dossier étudiant comme illustré ci - dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/view-folder-for-controllers.png)Afficher les dossiers pour les contrôleurs

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

## **vue Razor moteur:**

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:

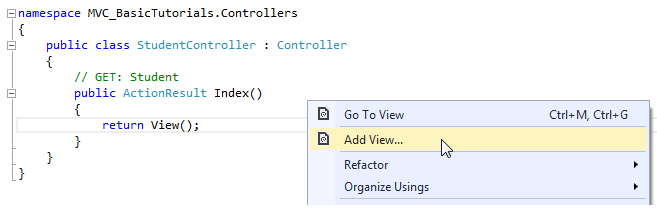
| **l'extension de fichier Voir** | **La description** |
| --- | --- |
| .cshtml | Vue C # Razor. Prise en charge de C # avec les balises HTML. |
| .vbhtml | Vue Visual Basic Razor. Prise en charge de Visual Basic avec des balises HTML. |
| .aspx | formulaire web ASP.Net |
| .ascx | contrôle Web ASP.NET |

En savoir [Razor syntaxe](http://www.tutorialsteacher.com/mvc/razor-syntax) dans la section suivante. Voyons voir comment créer une nouvelle vue à l'aide de Visual Studio 2013 pour Web avec MVC 5.

## **Créer une nouvelle Voir:**

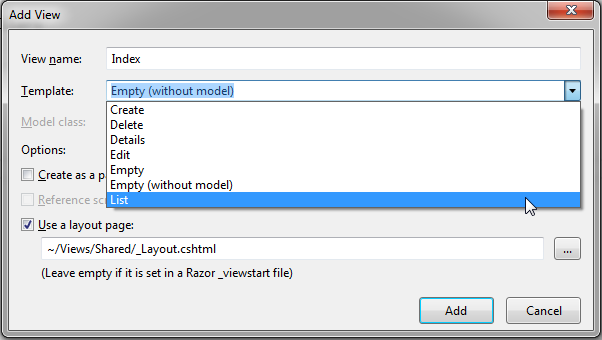
Nous avons déjà créé StudentController et le modèle de l'élève dans la section précédente.Maintenant, nous allons créer une vue de l'élève et de comprendre comment utiliser le modèle en vue.

Nous allons créer une vue, qui sera rendu à partir Index méthode StudentContoller. Alors, ouvrez une classe StudentController -> clic droit à l' intérieur de la méthode Index -> cliquez sur **Ajouter unevue ..**

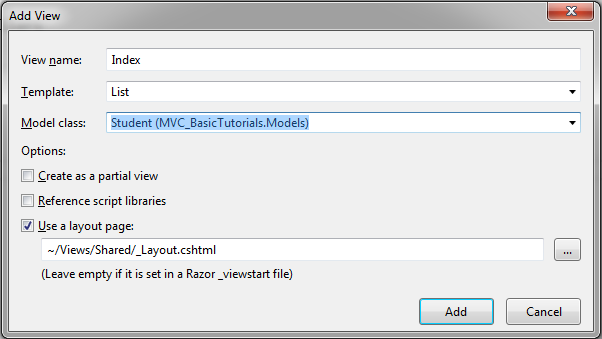
[](http://www.tutorialsteacher.com/Content/images/mvc/add-view-1.png)Créer une vue

Dans la boîte de dialogue Ajouter une vue, garder le nom de la vue comme indice. Il est bon de conserver le nom de vue le même que le nom de la méthode d'action afin que vous ne devez pas spécifier le nom de vue explicitement dans la méthode d'action tout en retournant la vue.

Sélectionnez le modèle d'échafaudage. Modèle de liste déroulante affiche les modèles par défaut disponibles pour créer, supprimer, Détails, Modifier, Liste ou vide. Sélectionnez "Liste" modèle parce que nous voulons montrer la liste des étudiants dans la vue.

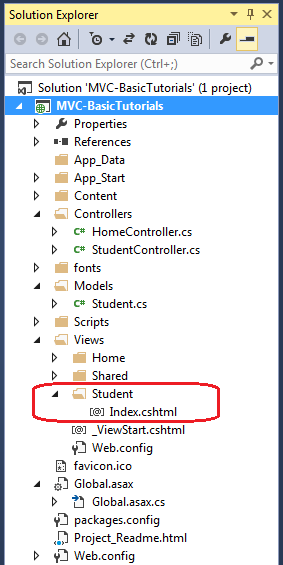
[](http://www.tutorialsteacher.com/Content/images/mvc/addview2.png)Vue

Maintenant, sélectionnez l'élève du dropdrown de classe Model. Modèle déroulant de classe affiche automatiquement le nom de toutes les classes dans le dossier du modèle. Nous avons déjà créé des étudiants de classe modèle dans la section précédente, de sorte qu'il serait inclus dans la liste déroulante.

[](http://www.tutorialsteacher.com/Content/images/mvc/addview3.png)Vue

Cochez la case "Utiliser une mise en page" case à cocher et sélectionnez la page \_Layout.cshtml pour cette vue, puis cliquez sur **Ajouter** bouton. Nous verrons plus loin ce qui est de mise en page ,mais pour pensons maintenant comme une page principale dans MVC.

Cela va créer vue Index sous Affichage - dossier> étudiant comme indiqué ci-dessous:

[](http://www.tutorialsteacher.com/Content/images/mvc/addview4.png)Vue

Le code suivant montre un Index.cshtml créé ci-dessus.

Views \ Student \ Index.cshtml:

@model IEnumerable<MVC\_BasicTutorials.Models.Student>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Index</h2>

<p>

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

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.StudentName)

</th>

<th>

@Html.DisplayNameFor(model => model.Age)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.StudentName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Age)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.StudentId }) |

@Html.ActionLink("Details", "Details", new { id=item.StudentId }) |

@Html.ActionLink("Delete", "Delete", new { id = item.StudentId })

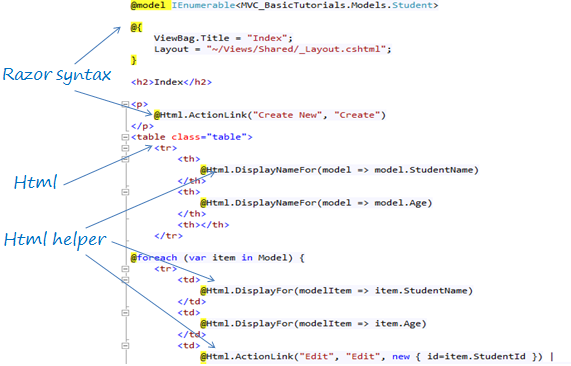
</td>

</tr>

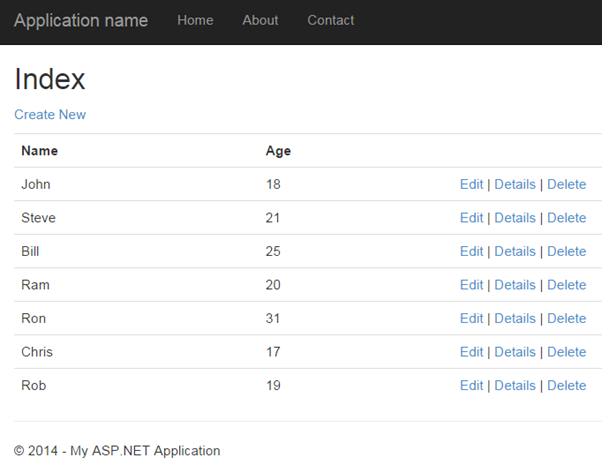
}

</table>

Comme vous pouvez le voir dans la vue ci-dessus de l'indice, il contient à la fois Html et codes de rasoir. expression de rasoir Inline commence par le symbole @. @Html Est une classe d'aide pour générer des commandes html. Vous apprendrez rasoir syntaxe et les aides html dans les sections à venir.

[](http://www.tutorialsteacher.com/Content/images/mvc/razor-view.png)Index.cshtml

La vue ci-dessus de l'indice ressemblerait ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/index-view.png)Index Vue

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



### **Points à retenir:**

1. View est une interface utilisateur qui affiche des données et gère l'interaction de l'utilisateur.
2. dossier Vues contient dossier séparé pour chaque contrôleur.
3. ASP.NET MVC supporte le moteur de vue Razor en plus de moteur de .aspx traditionnel.
4. vue Razor fichiers a l'extension .cshtml ou .vbhtml.

Apprenez à intégrer le modèle, la vue et le contrôleur dans la section suivante.

Intégrez Controller, Vue et modèle:

Nous avons déjà créé StudentController, le modèle et la vue dans les sections précédentes, mais on n'a pas intégré tous ces composants dans l'ordre de l'exécuter.

Le code suivant montre vue StudentController et Étudiant modèle de classe et créé dans les sections précédentes.

Exemple: StudentController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.[Mvc](http://www.tutorialsteacher.com/mvc/integrate-controller-view-model);

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

{

return View();

}

}

}

Exemple: Student classe Model

namespace MVC\_BasicTutorials.Models

{

public class Student

{

public int StudentId { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

}

Exemple: Index.cshtml pour afficher la liste des étudiants

@model IEnumerable<MVC\_BasicTutorials.Models.Student>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Index</h2>

<p>

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

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.StudentName)

</th>

<th>

@Html.DisplayNameFor(model => model.Age)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.StudentName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Age)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.StudentId }) |

@Html.ActionLink("Details", "Details", new { id=item.StudentId }) |

@Html.ActionLink("Delete", "Delete", new { id = item.StudentId })

</td>

</tr>

}

</table>

Maintenant, pour l'exécuter avec succès, nous avons besoin de passer un objet de modèle de contrôleur à la vue Index. Comme vous pouvez le voir dans le Index.cshtml ci-dessus, il utilise IEnumerable de l'étudiant comme un objet de modèle. Nous avons donc besoin de passer IEnumerable du modèle à partir de la méthode de la classe StudentController Indice d'action de l'élève, comme indiqué ci-dessous.

Exemple: Passing Modèle de contrôleur

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

{

var studentList = new List<Student>{

new Student() { StudentId = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentId = 2, StudentName = "Steve", Age = 21 } ,

new Student() { StudentId = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentId = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentId = 5, StudentName = "Ron" , Age = 31 } ,

new Student() { StudentId = 4, StudentName = "Chris" , Age = 17 } ,

new Student() { StudentId = 4, StudentName = "Rob" , Age = 19 }

};

// Get the students from the database in the real application

return View(studentList);

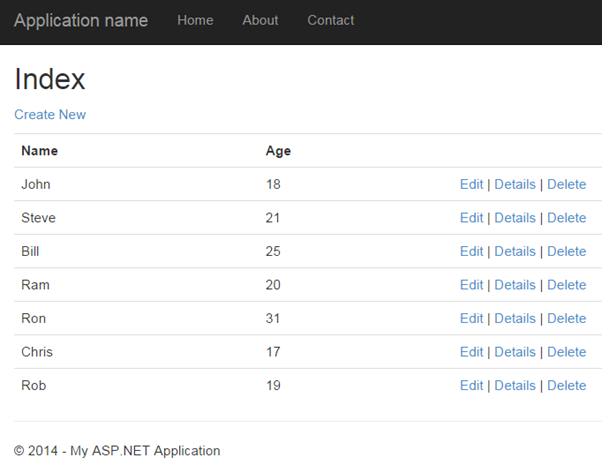
}

}

[Essayez-le](http://www.tutorialsteacher.com/codeeditor?cid=mvc-Dioxf9)

Comme vous pouvez le voir dans le code ci-dessus, nous avons créé une liste d'objets d'étudiants pour un exemple fin (dans la vraie vie applicatoin, vous pouvez chercher dans la base de données).Nous passons alors cet objet de liste en tant que paramètre dans la méthode View (). La méthode View () est définie dans la classe de contrôleur de base, qui se lie automatiquement objet de modèle à la vue.

Maintenant, vous pouvez exécuter le [MVC](http://www.tutorialsteacher.com/mvc/integrate-controller-view-model) projet en appuyant sur F5 et accédez à*http: // localhost / étudiant* . Vous verrez ci - dessous vue dans le navigateur.

[](http://www.tutorialsteacher.com/Content/images/mvc/index-view.png)Index Vue

Apprenez rasoir syntaxe dans la section suivante.

# [Razor Syntax](http://www.tutorialsteacher.com/mvc/razor-syntax):

Razor is one of the view engine supported in [ASP.NET MVC](http://www.tutorialsteacher.com/mvc/razor-syntax). Razor allows you to write mix of HTML and server side code using C# or Visual Basic. Razor view with visual basic syntax has .vbhtml file extension and C# syntax has .cshtml file extension.

[Razor syntax](http://www.tutorialsteacher.com/mvc/razor-syntax) has following Characteristics:

* **Compact**: [Razor syntax](http://www.tutorialsteacher.com/mvc/razor-syntax) is compact which enables you to minimize number of characters and keystrokes required to write a code.
* **Easy to Learn**: [Razor syntax](http://www.tutorialsteacher.com/mvc/razor-syntax) is easy to learn where you can use your familiar language C# or Visual Basic.
* **Intellisense**: [Razor syntax](http://www.tutorialsteacher.com/mvc/razor-syntax) supports statement completion within Visual Studio.

Now, let's learn how to write razor code.

## **Inline expression:**

Start with @ symbol to write server side C# or VB code with Html code. For example, write @Variable\_Name to display a value of a server side variable. For example, DateTime.Now returns a current date and time. So, write @DateTime.Now to display current datetime as shown below. A single line expression does not require a semicolon at the end of the expression.

C# [Razor Syntax](http://www.tutorialsteacher.com/mvc/razor-syntax):

<h1>[Razor syntax](http://www.tutorialsteacher.com/mvc/razor-syntax) demo</h1>

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

Result:

[**Razor syntax**](http://www.tutorialsteacher.com/mvc/razor-syntax) **demo**

08-09-2014

## **Multi-statement Code block:**

You can write multiple line of server side code enclosed in braces @{ ... }. Each line must ends with semicolon same as C#.

C# [Razor Syntax](http://www.tutorialsteacher.com/mvc/razor-syntax):

@{

var date = DateTime.Now.ToShortDateString();

var message = "Hello World";

}

<h2>Today's date is: @date </h2>

<h3>@message</h3>

Result:

[**Razor syntax**](http://www.tutorialsteacher.com/mvc/razor-syntax) **demo**

Today's date is: 08-09-2014

Hello World!

## **Display text from code block:**

Use @: or <text>/<text> to display texts within code block.

C# [Razor Syntax](http://www.tutorialsteacher.com/mvc/razor-syntax):

@{

var date = DateTime.Now.ToShortDateString();

string message = "Hello World!";

@:Today's date is: @date <br />

@message

}

Result:

[**Razor syntax**](http://www.tutorialsteacher.com/mvc/razor-syntax) **demo**

Today's date is: 08-09-2014

Hello World!

Display text using <text> within a code block as shown below.

[Razor Syntax](http://www.tutorialsteacher.com/mvc/razor-syntax):

@{

var date = DateTime.Now.ToShortDateString();

string message = "Hello World!";

<text>Today's date is:</text> @date <br />

@message

}

Result:

[**Razor syntax**](http://www.tutorialsteacher.com/mvc/razor-syntax) **demo**

Today's date is: 08-09-2014

Hello World!

## **if-else condition:**

Write if-else condition starting with @ symbol. The if-else code block must be enclosed in braces { }, even for single statement.

[Razor Syntax](http://www.tutorialsteacher.com/mvc/razor-syntax):

@if(DateTime.IsLeapYear(DateTime.Now.Year) )

{

@DateTime.Now.Year @:is a leap year.

}

else {

@DateTime.Now.Year @:is not a leap year.

}

Result:

2014 is not a leap year.

## **for loop:**

[Razor Syntax](http://www.tutorialsteacher.com/mvc/razor-syntax):

@for (int i = 0; i < 5; i++) {

@i.ToString() <br />

}

Result:

0

1

2

3

4

## **Model:**

Use @model to use model object anywhere in the view.

C# [Razor Syntax](http://www.tutorialsteacher.com/mvc/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>

Result:

**Student Detail:**

- Student Id: 1

- Student Name: John

- Age: 18

## **Declare Variables:**

Declare a variable in a code block enclosed in brackets and then use those variables inside html with @ symbol.

C# [Razor Syntax](http://www.tutorialsteacher.com/mvc/razor-syntax):

@{

string str = "";

if(1 > 0)

{

str = "Hello World!";

}

}

<p>@str</p>

Result:

Hello World!

So this was some of the important razor syntaxes. Visit asp.net to learn [razor syntax](http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-(c)) in detail.



### **Points to Remember :**

1. Use @ to write server side code.
2. Server side code block starts with @{\* code \* }
3. Use @: or <text></<text> to display text from code block.
4. if condition starts with @if{ }
5. for loop starts with @for
6. @model allows you to use model object anywhere in the view.

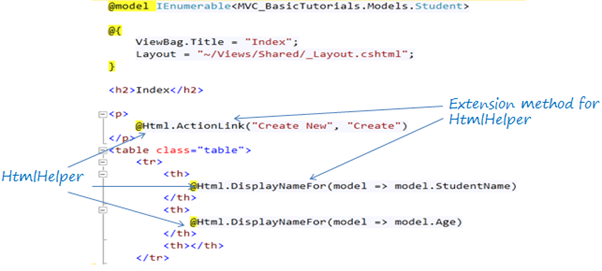
Learn how to use HtmlHelpers in razor view in the next section.

# HTML Helpers:

In this section, you will learn what are Html helpers and how to use them in the razor view.

HtmlHelper class generates html elements using the model class object in razor view. It binds the model object to html elements to display value of model properties into html elements and also assigns the value of the html elements to the model properties while submitting web form. So always use HtmlHelper class in razor view instead of writing html tags manually.

The following figure shows the use of HtmlHelper class in the razor view.

[](http://www.tutorialsteacher.com/Content/images/mvc/htmlhelpers.png)HTML Helpers

As you can see in the above figure, **@Html** is an object of HtmlHelper class . (@ symbol is used to access server side object in razor syntax). Html is a property of type HtmlHelper included in base class of razor view WebViewPage. ActionLink() and DisplayNameFor() is extension methods included in HtmlHelper class.

HtmlHelper class generates html elements. For example, @Html.ActionList("Create New", "Create")would generate anchor tag <a href="/Student/Create">Create New</a>.

There are many [extension methods for HtmlHelper](http://msdn.microsoft.com/en-us/library/system.web.mvc.html(v=vs.118).aspx) class, which creates different html controls.

The following table lists HtmlHelper methods and html control each method generates.

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

The difference between calling the HtmlHelper methods and using an html tags is that the HtmlHelper method is designed to make it easy to bind to view data or model data.



### **Points to Remember :**

1. HtmlHelper extension method generates html elements based on model properties.
2. It is advisable to use "For" extension methods for compile time type checking e.g. TextBoxFor, EditorFor, CheckBoxFor etc.

Learn about various HtmlHelper methods in the next few sections.

# Créer TextBox en utilisant HtmlHelper:

Apprenez à créer un contrôle textbox utilisant HtmlHelper en vue de rasoir dans cette section.

classe HtmlHelper comprend deux méthodes d'extension qui crée une zone de texte (<input type = "text">) élément en vue rasoir: TextBox () et TextBoxFor (). La méthode TextBox () est vaguement méthode dactylographié alors TextBoxFor () est une méthode fortement typée.

Nous allons utiliser le modèle de l'étudiant qui suit avec TextBox () et la méthode TextBoxFor ().

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public bool isNewlyEnrolled { get; set; }

public string Password { get; set; }

}

## **Zone de texte():**

La méthode Html.TextBox () crée <input type = "text"> avec les attributs nom spécifié, la valeur et html.

### **TextBox () signature de la méthode:**

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

Méthode de TextBox a beaucoup de surcharges. S'il vous plaît visitez MSDN de connaître toutes les[surcharges de TextBox (méthode)](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.textbox(v=vs.118).aspx) .

La méthode TextBox () est une méthode faiblement typé, car le nom du paramètre est une chaîne.Le paramètre de nom peut être un nom de propriété de l'objet modèle. Il se lie la propriété spécifiée avec textbox. Donc, il affiche automatiquement une valeur de la propriété de modèle dans une zone de texte et vice-versa.

Exemple: Html.TextBox () dans Razor Voir

@model Student

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

Résultat Html:

<input class="form-control"

id="StudentName"

name="StudentName"

type="text"

value="" />

Dans l'exemple ci-dessus, le premier paramètre est "StudentName" propriété de l'étudiant classe modèle qui sera défini comme un nom & id de textbox. Le second paramètre est une valeur à afficher dans une zone de texte, qui est nulle dans l'exemple ci-dessus parce que la méthode TextBox () sera automatiquement afficher une valeur de la propriété StudentName dans la zone de texte. Le troisième paramètre sera défini comme attribut de classe. paramètre htmlAttributes est un type d'objet, il peut donc être objet anonyme et les attributs nom sera ses propriétés commençant par le symbole @.

Vous pouvez également spécifier un nom pour la zone de texte. Cependant, il ne sera pas se lier à un modèle.

Exemple: Html.TextBox () dans Razor Voir

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

Résultat Html:

<input class="form-control"

id="myTextBox"

name="myTextBox"

type="text"

value="This is value" />

[Sortie dans le navigateur](http://www.tutorialsteacher.com/Content/images/mvc/htmlhelper-textbox.png)Sortie de TextBox () Méthode Html Helper

## **TextBoxFor:**

helper TextBoxFor est une méthode d'extension fortement typé. Il génère un élément de saisie de texte pour la propriété de modèle spécifiée en utilisant une expression lambda. TextBoxFor méthode se lie à une propriété de l'objet modèle spécifié pour saisir du texte. Donc, il affiche automatiquement une valeur de la propriété de modèle dans une zone de texte et vice-versa.

### **TextBoxFor () Méthode Signature:**

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

Visitez MSDN de connaître toutes les [surcharges de méthode TextBoxFor ()](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.textboxfor(v=vs.118).aspx) .

Exemple: TextBoxFor () dans Razor Voir

@model Student

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

Résultat Html:

<input class="form-control"

id="StudentName"

name="StudentName"

type="text"

value="John" />

Dans l'exemple ci-dessus, le premier paramètre dans la méthode TextBoxFor () est une expression lambda qui spécifie la propriété StudentName à se lier à la zone de texte. Il génère un élément de saisie de texte avec id & nom défini au nom de la propriété. L'attribut de valeur sera définie à la valeur d'une propriété de StudentName par exemple John. La figure suivante montre l'élément de saisie de texte genered par exemple ci-dessus.

[http://www.tutorialsteacher.com/Content/images/mvc/textboxfor-helpermethod.png](http://www.tutorialsteacher.com/Content/images/mvc/textboxfor-helpermethod.png)Sortie de TextBoxFor () Méthode Html Helper

## **Différence entre TextBox et TextBoxFor:**

* @ Html.TextBox () est vaguement méthode alors tapé @ 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. Il lancera courir exception de temps.
* 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.(Vue fourni ne compile au moment de l'exécution.)

# Créer TextArea utilisant HtmlHelper:

Apprenez à créer un contrôle TextArea utilisant HtmlHelper en vue de rasoir dans cette section.

classe HtmlHelper comprend deux méthodes d'extension pour générer une ligne multiple <textarea> élément dans une vue de rasoir: TextArea () et TextAreaFor (). Par défaut, il crée textarea avec des rangées = 2 et cols = 20.

Nous allons utiliser le modèle de l'étudiant qui suit avec le TextArea () et la méthode TextAreaFor ().

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public string Description { get; set; }

}

## **TextArea ():**

La méthode Html.TextArea () crée <lignes textarea = "2" cols = "20"> élément avec des attributs de nom, valeur et html spécifiés.

### **TextBox () Méthode Signature:**

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

Méthode TextArea a beaucoup de surcharges. S'il vous plaît visitez MSDN de connaître toutes les[surcharges de méthode TextArea](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.textareaextensions(v=vs.118).aspx) .

La méthode TextArea () est une méthode sans typage car le paramètre de nom est une chaîne. Le paramètre de nom peut être un nom de propriété de l'objet modèle. Il se lie à une propriété spécifiée avec la zone de texte. Donc, il affiche automatiquement une valeur de la propriété de modèle dans une zone de texte et vice-versa.

Exemple: Html.TextArea () dans Razor Voir

@model Student

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

Résultat Html:

<textarea class="form-control"

id="Description"

name="Description"

rows="2"

cols="20">This is value</textarea>

Dans l'exemple ci-dessus, le premier paramètre est la propriété «Description» de l'étudiant classe modèle qui sera défini comme un nom & id de textarea. Le second paramètre est une valeur à afficher dans un textarea, qui est nulle dans l'exemple ci-dessus parce que la méthode TextArea () affichera automatiquement une valeur de la propriété Description dans le textarea. Le troisième paramètre sera défini comme attribut de classe. paramètre htmlAttributes est un type d'objet, il peut donc être objet anonyme et les attributs nom sera ses propriétés commençant par le symbole @.

Vous pouvez aussi spécifier un nom pour le textarea. Cependant, il ne sera pas lié à un modèle.

Exemple: Html.TextArea () dans Razor Voir

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

Résultat Html:

<textarea class="form-control"

cols="20"

id="myTextArea"

name="myTextArea"

rows="2">This is value</textarea>

L'exemple ci-dessus générer des éléments d'entrée comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/textarea-htmlhelper.png)Sortie de TextArea () Méthode d'assistance

## **TextAreaFor:**

helper TextAreaFor est une méthode d'extension fortement typé. Il génère une ligne de plusieurs élément <textarea> pour la propriété de l'objet de modèle spécifié en utilisant une expression lambda. TextAreaFor méthode se lie à une propriété de l'objet modèle spécifié à l'élément textarea.Donc, il affiche automatiquement une valeur de la propriété de modèle dans un textarea et vice-versa.

### **TextBoxFor () Méthode Signature:**

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

Visitez MSDN de connaître toutes les [surcharges de TextAreaFor ()](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.textareaextensions(v=vs.118).aspx) .

Exemple: TextAreaFor () dans Razor Voir

@model Student

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

Résultat Html:

<textarea class="form-control"

cols="20"

id="Description"

name="Description"

rows="2"></textarea>

Dans l'exemple ci-dessus, le premier paramètre dans la méthode TextAreaFor () est une expression lambda qui spécifie la propriété de modèle pour être lié à l'élément textarea. Nous avons spécifié Description de la propriété dans l'exemple ci-dessus. Donc, il génère <textarea> élément avec id & name défini sur le nom de la propriété - Description. La valeur de textarea sera réglée sur la valeur d'une propriété Description.

# Créer CheckBox utilisant HtmlHelper:

Apprenez à créer un contrôle de case à cocher en utilisant HtmlHelper en vue de rasoir dans cette section.

classe HtmlHelper comprend deux méthodes d'extension pour générer un <input type = "checkbox"> élément en vue rasoir: CheckBox () et CheckBoxFor ().

Nous allons utiliser le modèle de l'étudiant qui suit avec CheckBox () et CheckBoxFor () méthode.

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public bool isNewlyEnrolled { get; set; }

public string Password { get; set; }

}

## **CheckBox ():**

Le Html.CheckBox () est une méthode faiblement typé qui génère un <input type = "checkbox"> avec le nom spécifié, isChecked attributs booléens et html.

### **CheckBox () Méthode Signature:**

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

S'il vous plaît visitez MSDN de connaître toutes les [surcharges de CheckBox () méthode](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.checkbox(v=vs.118).aspx) .

Exemple: Html.CheckBox () dans Razor Voir

@Html.CheckBox("isNewlyEnrolled", true)

Résultat Html:

<input checked="checked"

id="isNewlyEnrolled"

name="isNewlyEnrolled"

type="checkbox"

value="true" />

Dans l'exemple ci-dessus, le premier paramètre est "isNewlyEnrolled" propriété de l'étudiant classe modèle qui sera défini comme un nom & id de textbox. Le second paramètre est une valeur booléenne, qui vérifie ou décoche la case à cocher.

## **CheckBoxFor:**

helper CheckBoxFor est une méthode d'extension fortement typé. Il génère <input type = "checkbox"> pour la propriété de modèle spécifié en utilisant une expression lambda. méthode CheckBoxFor se lie à une propriété de l'objet modèle spécifié pour checkbox élément. Donc, il vérifie automatiquement ou non coché une case à cocher en fonction de la valeur de la propriété.

### **CheckBoxFor () Méthode Signature:**

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

Visitez MSDN de connaître toutes les [surcharges de méthode CheckBoxFor ()](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.checkboxfor(v=vs.118).aspx) .

Exemple: Html.CheckBoxFor () dans Razor Voir

@model Student

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

Résultat Html:

<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" />

Dans l'exemple ci-dessus, le premier paramètre dans la méthode CheckBoxFor () est une expression lambda qui spécifie la propriété de modèle pour être lié avec l'élément de case à cocher. Nous avons spécifié la propriété isNewlyEnrolled dans l'exemple ci-dessus. Donc, il génère <input type = "checkbox"> avec id & name défini sur le nom de la propriété - isNewlyEnrolled. L'attribut de valeur sera définie à la valeur d'une propriété booléenne isNewlyEnrolled.

Dans le résultat de Html ci-dessus, notez qu'il a généré champ caché supplémentaire avec le même nom et la valeur = false. En effet, lorsque vous soumettez un formulaire avec une case à cocher, la valeur est uniquement affichée si la case est cochée. Donc, si vous laissez la case cochée alors rien ne sera envoyé au serveur lorsque dans de nombreuses situations que vous voudriez faux être envoyé à la place. Comme l'entrée cachée a le même nom que la case à cocher, puis, si la case est cochée, vous aurez encore un «faux» envoyé au serveur.

# Créer RadioButton en utilisant HtmlHelper:

Apprenez à créer un contrôle de bouton radio en utilisant HtmlHelper en vue de rasoir dans cette section.

classe HtmlHelper comprennent deux méthodes d'extension pour générer un <input type = "radio"> dans une vue de rasoir: RadioButton () et RadioButtonFor ().

Nous allons utiliser le modèle de l'élève suivant la méthode RadioButton () et RadioButtonFor ().

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public string Gender { get; set; }

}

## **Bouton radio():**

La méthode Html.RadioButton () crée un élément de bouton radio avec un nom spécifié, booléen IsChecked et les attributs HTML.

### **RadioButton () Méthode Signature:**

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

S'il vous plaît visitez MSDN de connaître toutes les [surcharges de méthode RadioButton ()](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.RadioButton(v=vs.118).aspx) .

Exemple: Html.RadioButton () dans Razor Voir

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

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

Résultat Html:

Male: <input checked="checked"

id="Gender"

name="Gender"

type="radio"

value="Male" />

Female: <input id="Gender"

name="Gender"

type="radio"

value="Female" />

Dans l'exemple ci-dessus, nous avons créé deux boutons radio pour le «genre» propriété. Le second paramètre est une valeur qui sera envoyée au serveur, si le bouton radio respectif est cochée. Si le bouton radio Homme est sélectionnée, la valeur de chaîne "Homme" sera attribué à une propriété du modèle genre et soumis au serveur. L'exemple ci-dessus crée deux boutons radio comme indiqué ci-dessous.

[Sortie dans le navigateur](http://www.tutorialsteacher.com/Content/images/mvc/htmlhelper-radio.png)Sortie de RadioButton () Méthode

## **RadioButtonFor:**

helper RadioButtonFor est une méthode d'extension fortement typé. Il génère <input type = "radio"> pour la propriété spécifiée en utilisant une expression lambda. méthode RadioButtonFor lie un modèle objet propriété spécifiée au contrôle RadioButton. Donc, il coché ou décoché un RadioButton basé sur la valeur de la propriété automatiquement.

### **RadioButtonFor () Méthode Signature:**

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

Visitez MSDN de connaître toutes les [surcharges de RadioButtonFor ()](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.RadioButtonFor(v=vs.118).aspx) .

Exemple: Html.RadioButtonFor () dans Razor Voir

@model Student

@Html.RadioButtonFor(m => m.Gender,"Male")

@Html.RadioButtonFor(m => m.Gender,"Female")

Résultat Html:

<input checked="checked"

id="Gender"

name="Gender"

type="radio"

value="Male" />

<input id="Gender"

name="Gender"

type="radio"

value="Female" />

Dans l'exemple ci-dessus, le premier paramètre dans la méthode RadioButtonFor () est une expression lambda qui spécifie la propriété de modèle à lier avec l'élément RadioButton. Nous avons créé deux boutons radio pour la propriété de genre dans l'exemple ci-dessus. Donc, il génère deux <input type = "RadioButton"> avec id & name défini sur le nom de la propriété - Sexe. Le second paramètre est une valeur qui sera envoyée au serveur lorsque le formulaire sera soumis.

# Créer DropDownList en utilisant HtmlHelper:

Apprenez à générer le contrôle dropdownlist utilisant HtmlHelper en vue de rasoir dans cette section.

HtmlHelper classe comprend deux méthodes d'extension pour générer un élément <select> dans une vue de rasoir: DropDownList () et DropDownListFor ().

Nous allons utiliser le modèle de l'élève qui suit avec DropDownList () et la méthode DropDownListFor ().

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public Gender StudentGender { get; set; }

}

public enum Gender

{

Male,

Female

}

## **La liste déroulante():**

La méthode Html.DropDownList () génère un élément de sélection avec le nom spécifié, les éléments de liste et les attributs HTML.

### **DropDownList () signature de la méthode:**

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

S'il vous plaît visitez MSDN de connaître toutes les [surcharges de méthode DropDownList ()](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.selectextensions.dropdownlist(v=vs.118).aspx) .

Exemple: Html.DropDownList () dans Razor Voir

@model Student

@Html.DropDownList("StudentGender",

new SelectList(Enum.GetValues(typeof(Gender))),

"Select Gender",

new { @class = "form-control" })

Résultat Html:

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

<option>Select Gender</option>

<option>Male</option>

<option>Female</option>

</select>

Dans l'exemple ci - dessus, le premier paramètre est un nom de propriété pour laquelle nous voulons afficher les éléments de liste. Le deuxième paramètre est la liste des valeurs à inclure dans le dropdownlist. Nous avons utilisé [Enum](http://www.tutorialsteacher.com/csharp/csharp-enum) méthodes pour obtenir les valeurs enum de genre. Le troisième paramètre est une étiquette qui sera la première liste élément et le quatrième paramètre est pour les attributs HTML comme css à appliquer sur le dropdownlist.

## **DropDownListFor:**

helper DropDownListFor est une méthode d'extension fortement typé. Il génère <select> pour la propriété spécifiée en utilisant une expression lambda. DropDownListFor méthode se lie à une propriété de l'objet modèle spécifié pour DropDownList contrôle. Donc la liste automatiquement les articles dans DropDownList en fonction de la valeur de la propriété.

### **DropDownListFor () signature de la méthode:**

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

Visitez MSDN de connaître toutes les [surcharges de DropDownListFor ()](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.DropDownListFor(v=vs.118).aspx) .

L'exemple suivant crée la liste déroulante pour le genre enum ci-dessus.

Exemple: DropDownListFor () dans Razor Voir

@model Student

@Html.DropDownListFor(m => m.StudentGender,

new SelectList(Enum.GetValues(typeof(Gender))),

"Select Gender")

Résultat Html:

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

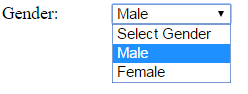
<option>Select Gender</option>

<option>Male</option>

<option>Female</option>

</select>

Dans l'exemple ci-dessus, le premier paramètre dans la méthode DropDownListFor () est une expression lambda qui spécifie la propriété de modèle à se lier avec l'élément select. Nous avons spécifié la propriété StudentGender de type enum. Le deuxième paramètre spécifie les éléments à afficher dans la liste déroulante en utilisant SelectList. Le troisième paramètre est optionLabel qui sera le premier élément de dropdownlist. Alors maintenant, il génère élément <select> avec id & name défini sur le nom de la propriété - StudentGener et deux éléments de la liste - Masculin et féminin comme illustré ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/htmlhelper-dropdown.png)Méthode de sortie de DropDownList () ou DropDownListFor ()

# Créer champ caché en utilisant HtmlHelper:

Apprenez à générer champ caché en utilisant HtmlHelper en vue de rasoir dans cette section.

classe HtmlHelper comprend deux méthodes d'extension pour générer un champ caché (<input type = "hidden">) élément dans une vue de rasoir: Hidden () et HiddenFor ().

Nous allons utiliser le modèle de l'élève qui suit avec Caché () et la méthode HiddenFor ().

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public bool isNewlyEnrolled { get; set; }

public string Password { get; set; }

}

## **Caché():**

La méthode Html.Hidden () génère un élément de champ caché d'entrée avec des attributs de nom, valeur et html spécifiés.

### **() méthode cachée signature:**

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

Méthode cachée () a beaucoup de surcharges. S'il vous plaît visitez MSDN de connaître toutes les[surcharges de méthode cachée ()](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.hidden(v=vs.118).aspx) .

L'exemple suivant crée un champ caché de la propriété StudentID du modèle étudiant. Il se lie StudentID avec le champ caché, de sorte qu'il peut attribuer une valeur de StudentID au champ caché et vice-versa.

Exemple: Html.Hidden () dans Razor Voir

@model Student

@Html.Hidden("StudentId")

Résultat Html:

<input id="StudentId"

name="StudentId"

type="hidden"

value="1" />

## **HiddenFor:**

HiddenFor méthode d'aide est une méthode d'extension fortement typé. Il génère un élément d'entrée cachée pour la propriété de modèle spécifiée en utilisant une expression lambda. HiddenFor méthode se lie à une propriété de l'objet modèle spécifié pour <input type = "hidden">. Ainsi, il définit automatiquement une valeur de la propriété de modèle à champ caché et vice-versa.

### **HiddenFor () signature de la méthode:**

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

Visitez MSDN de connaître toutes les [surcharges de méthode HiddenFor ()](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.hiddenfor(v=vs.118).aspx) .

Exemple: HiddenFor () dans Razor Voir

@model Student

@Html.HiddenFor(m => m.StudentId)

Résultat Html:

<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" />

Dans l'exemple ci-dessus, le premier paramètre dans la méthode HiddenFor () est une expression lambda qui spécifie la propriété de modèle à lier avec le champ caché. Nous avons spécifié la propriété StudentID dans l'exemple ci-dessus. Donc, il génère entrée élément de texte avec id & name défini sur le nom de la propriété. L'attribut de valeur sera définie à la valeur d'une propriété StudentID qui est 1 dans l'exemple ci-dessus.

S'il vous plaît notez qu'il a créé de données attribut de html5 qui est utilisé pour la validation dans ASP.Net MVC.

# Créer champ Mot de passe en utilisant HtmlHelper:

Apprenez à générer champ Mot de passe en utilisant HtmlHelper en vue de rasoir dans cette section.

classe HtmlHelper comprend deux méthodes d'extension pour générer un champ de mot de passe (<input type = "password">) de l'élément dans une vue de rasoir: Mot de passe () et PasswordFor ().

Nous allons utiliser le modèle de l'élève suivant avec mot de passe () et la méthode PasswordFor ().

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public bool isNewlyEnrolled { get; set; }

public string OnlinePassword { get; set; }

}

## **Mot de passe():**

La méthode Html.Password () génère un élément de mot de passe d'entrée avec des attributs de nom, valeur et html spécifiés.

### **Mot de passe () signature de la méthode:**

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

Méthode Password () a beaucoup de surcharges. S'il vous plaît visitez MSDN de connaître toutes les[surcharges de la méthode de mot de passe ()](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.password(v=vs.118).aspx) .

Exemple: Html.Password () dans Razor Voir

@model Student

@Html.Password("OnlinePassword")

Résultat Html:

<input

id="OnlinePassword"

name="OnlinePassword"

type="password"

value="" />

L'exemple ci-dessus va créer champ de mot de passe pour la propriété "OnlinePassword" comme indiqué ci-dessous.

[Sortie dans le navigateur](http://www.tutorialsteacher.com/Content/images/mvc/htmlhelper-dropdown.png)Sortie de mot de passe () ou PasswordFor () Méthode

## **PasswordFor ():**

helper PasswordFor est une méthode d'extension fortement typé. Il génère un <input type = "password"> pour la propriété d'objet de modèle spécifié en utilisant une expression lambda.méthode PasswordFor se lie à une propriété de l'objet modèle spécifié pour <input type = "password">. Ainsi, il définit automatiquement une valeur de la propriété de modèle au champ de mot de passe et vice-versa.

### **PasswordFor () signature de la méthode:**

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

Visitez MSDN de connaître toutes les [surcharges de méthode PasswordFor ()](http://msdn.microsoft.com/en-us/library/system.web.mvc.html.inputextensions.passwordfor(v=vs.118).aspx) .

Exemple: PasswordFor () dans Razor Voir

@model Student

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

Résultat Html:

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

Dans l'exemple ci-dessus, le premier paramètre dans la méthode PasswordFor () est une expression lambda qui spécifie la propriété de modèle à se lier avec le champ de mot de passe. Nous avons spécifié la propriété de mot de passe dans l'exemple ci-dessus. Donc, il génère élément de mot de passe d'entrée avec id & nom défini au nom de la propriété. L'attribut de valeur sera définie à la valeur d'une propriété de mot de passe qui est "mypassword" dans l'exemple ci-dessus.

# Créer Html chaîne en utilisant HtmlHelper:

Apprenez à créer la chaîne html littérale utilisant HtmlHelper en vue de rasoir dans cette section.

classe HtmlHelper comprend deux méthodes d'extension pour générer chaîne HTML: Affichage () et DisplayFor ().

Nous allons utiliser le modèle de l'élève qui suit avec l'affichage () et la méthode DisplayFor ().

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

## **Afficher():**

Le Html.Display () est une méthode faiblement typé qui génère une chaîne en vue de rasoir pour la propriété spécifiée du modèle.

Affichage () Méthode Signature: MvcHtmlString Display(string expression)

Méthode d' affichage () a beaucoup de surcharges. S'il vous plaît visitez MSDN de connaître toutes les [surcharges de l' affichage () Méthode](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.displayextensions.display(v=vs.118).aspx)

Exemple: Html.Display () dans Razor Voir

@Html.Display("StudentName")

Résultat Html:

"Steve"

## **DisplayFor:**

DisplayFor méthode d'aide est une méthode d'extension fortement typé. Il génère une chaîne de html pour la propriété de l'objet modèle spécifié en utilisant une expression lambda.

DisplayFor () Méthode Signature: MvcHtmlString DisplayFor(<Expression<Func<TModel,TValue>> expression)

Visitez MSDN de connaître toutes les [surcharges de méthode DisplayFor ()](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.displayextensions.displayfor(v=vs.118).aspx) .

Exemple: PasswordFor () dans Razor Voir

@model Student

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

Résultat Html:

" Steve "

Dans l'exemple ci-dessus, nous avons spécifié la propriété StudentName du modèle étudiant en utilisant l'expression lambda dans la méthode DisplayFor (). Ainsi, il génère une chaîne html avec la valeur de la propriété StudentName, qui est "Steve" dans l'exemple ci-dessus.

# Créer l'étiquette en utilisant HtmlHelper:

Apprenez à créer <label> en utilisant HtmlHelper en vue de rasoir dans cette section.

classe HtmlHelper comprend deux méthodes d'extension pour générer l'étiquette html: Étiquette () et LabelFor ().

Nous allons utiliser suivant le modèle de l'étudiant avec la démo Étiquette () et la méthode LabelFor ().

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

}

## **Étiquette():**

La méthode Html.Label () génère un élément <label> pour une propriété spécifiée de l'objet modèle.

Étiquette () Méthode Signature: MvcHtmlString Label(string expression, string labelText, object htmlAttributes)

Méthode Étiquette () a beaucoup de surcharges. S'il vous plaît visitez MSDN de connaître toutes les[surcharges de label () Méthode](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.labelextensions.label(v=vs.118).aspx)

Exemple: Html.Label () dans Razor Voir

@Html.Label("StudentName")

Résultat Html:

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

Dans l'exemple ci - dessus, nous avons spécifié une propriété StudentName comme une chaîne.Donc, il va créer élément <label> à afficher *Nom* .

Vous pouvez spécifier un autre texte de l'étiquette au lieu du nom de la propriété, comme indiqué ci-dessous.

Exemple: Html.Label () dans Razor Voir

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

Résultat Html:

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

## **LabelFor:**

helper LabelFor est une méthode d'extension fortement typé. Il génère un élément label html pour la propriété de l'objet modèle spécifié en utilisant une expression lambda.

LabelFor () Méthode Signature: MvcHtmlString LabelFor(<Expression<Func<TModel,TValue>> expression)

Visitez MSDN de connaître toutes les [surcharges de méthode LabelFor ()](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.labelextensions.labelfor(v=vs.118).aspx) .

Exemple: LaeblFor () dans Razor Voir

@model Student

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

Résultat Html:

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

Dans l'exemple ci-dessus, nous avons spécifié la propriété StudentName du modèle étudiant en utilisant l'expression lambda dans la méthode LaebelFor (). Donc, il génère <label> et set texte de l'étiquette à la même chose que StudentName nom de la propriété.

# 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.

| **DataType de propriété** | **Html Element** |
| --- | --- |
| chaîne | <Input type = "text"> |
| int | <Input type = "Numéro"> |
| décimal, float | <Input type = "text"> |
| booléen | <Input type = "checkbox"> |
| Enum | <Input type = "text"> |
| DateTime | <Input type = "datetime"> |

Nous allons utiliser la classe de modèle suivant avec l'éditeur et la méthode EditorFor.

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public bool isNewlyEnrolled { get; set; }

public string Password { get; set; }

public DateTime DoB { get; set; }

}

## **Editeur():**

Editor () méthode nécessite un paramètre d'expression de chaîne pour spécifier le nom de la propriété. Il creats un élément HTML basé sur le type de données de la propriété spécifiée.

Editor () signature: MvcHtmlString Editor(string propertyname)

Visitez MSDN de connaître toutes les [surcharges de Editor () Méthode](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.editorextensions.editor(v=vs.118).aspx)

Prenons l'exemple suivant pour comprendre la méthode Editor ().

Exemple: Editor () en vue Razor

StudentId: @Html.Editor("StudentId")

Student Name: @Html.Editor("StudentName")

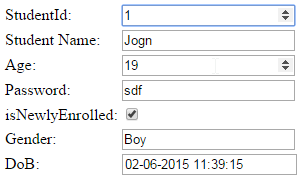
Age: @Html.Editor("Age")

Password: @Html.Editor("Password")

isNewlyEnrolled: @Html.Editor("isNewlyEnrolled")

Gender: @Html.Editor("Gender")

DoB: @Html.Editor("DoB")

[](http://www.tutorialsteacher.com/Content/images/mvc/htmlhelper-editor.png)Méthode de sortie de l'Éditeur () et EditorFor ()

Dans l'exemple ci-dessus, nous avons spécifié les noms de propriétés de modèle d'étudiant comme une chaîne. Ainsi, la méthode Editor () a créé les éléments d'entrée appropriées en fonction du type de données, comme indiqué dans la figure ci-dessus.

## **EditorFor:**

méthode EditorFor () est une méthode fortement typée. Elle exige l'expression lambda pour spécifier une propriété de l'objet modèle.

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

Visitez MSDN de connaître toutes les [surcharges de EditorFor () Méthode](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.editorextensions.editorfor(v=vs.118).aspx)

Exemple: EditorFor () en vue Razor

StudentId: @Html.EditorFor(m => m.StudentId)

Student Name: @Html.EditorFor(m => m.StudentName)

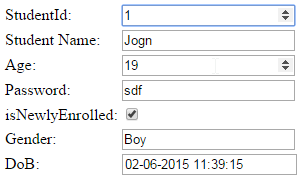
Age: @Html.EditorFor(m => m.Age)

Password: @Html.EditorFor(m => m.Password)

isNewlyEnrolled: @Html.EditorFor(m => m.isNewlyEnrolled)

Gender: @Html.EditorFor(m => m.Gender)

DoB: @Html.EditorFor(m => m.DoB)

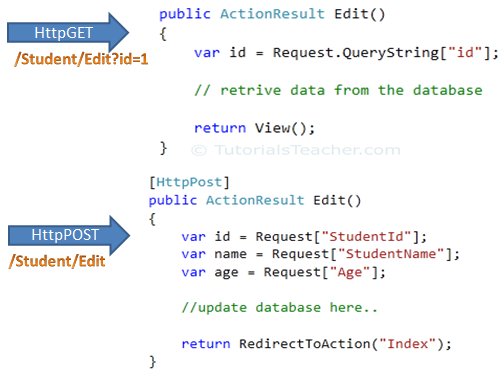
[](http://www.tutorialsteacher.com/Content/images/mvc/htmlhelper-editor.png)Méthode de sortie de l'Éditeur () et EditorFor ()

Dans l'exemple ci-dessus de la méthode EditorFor (), nous avons spécifié le nom de propriété en utilisant l'expression lambda. Le résultat serait le même que la méthode Editor (), comme indiqué dans la figure ci-dessus.

# Modèle Reliure:

Dans cette section, vous apprendrez sur la liaison modèle framework MVC.

Pour comprendre la liaison dans MVC modèle, d'abord nous allons voir comment vous pouvez obtenir les valeurs de demande de http dans la méthode d'action en utilisant le style de ASP.NET traditionnel.La figure suivante montre comment vous pouvez obtenir les valeurs de HttpGet et HttpPost demande en utilisant l'objet de demande directement dans la méthode d'action.

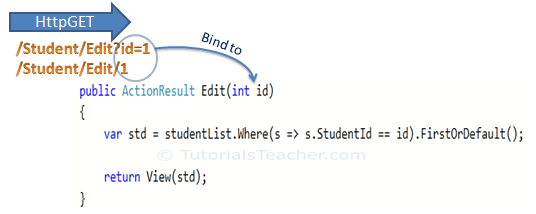
[](http://www.tutorialsteacher.com/Content/images/mvc/request-data.png)Accès Demande de données

Comme vous pouvez le voir dans la figure ci-dessus, nous utilisons le Request.QueryString et demande (Request.Form) objet pour obtenir la valeur de HttpGet et HttpPost demande. Accès à des valeurs de demande à l'aide de l'objet Request est une activité de dépérissement lourd et le temps.

Avec la liaison de modèle, framework MVC convertit les valeurs de demande http (de chaîne de requête ou d'une collection de formulaire), les paramètres de méthode d'action. Ces paramètres peuvent être de type primitif ou d'un type complexe.

## **Reliure type primitif:**

HttpGet demande intègre des données dans une chaîne de requête. framework MVC convertit automatiquement une chaîne de requête aux paramètres de la méthode d'action. Par exemple, la chaîne de requête "id" dans la requête GET suivante serait automatiquement mappé au paramètre id de la méthode d'action Edit ().

[](http://www.tutorialsteacher.com/Content/images/mvc/model-binding-1.png)modèle de liaison

http://www.tutorialsteacher.com/Content/images/tips.pngCette liaison est insensible à la casse. Donc paramètre "id" peut être "ID" ou "Id".

Vous pouvez également avoir plusieurs paramètres dans la méthode d'action avec les différents types de données. les valeurs de chaîne de requête seront converties en paramters basé sur le nom correspondant.

Par exemple, *http: // localhost / Etudiant / Modifier id = 1 & name = John?* Serait mapper id et lenom du paramètre de la méthode Edit action suivante.

Exemple: Convertir QueryString à la méthode d'action Paramètres

public ActionResult Edit(int id, string name)

{

// do something here

return View();

}

## **Reliure type complexe:**

liaison aussi Modèle fonctionne sur les types complexes. contraignant framework MVC Modèle convertit automatiquement les données de champ de formulaire de demande de HttpPost les propriétés d'un paramètre de type complexe d'une méthode d'action.

Considérez les classes de modèles suivants.

classes de modèle - C #:

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public Standard standard { get; set; }

}

public class Standard

{

public int StandardId { get; set; }

public string StandardName { get; set; }

}

Maintenant, vous pouvez créer une méthode d'action qui comprend le type d'étudiant paramètre.Dans l'exemple suivant, Edit méthode d'action (HttpPost) comprend le type d'étudiant paramètre.

méthode d'action avec le paramètre de type complexe Exemple:

[HttpPost]

public ActionResult Edit(Student std)

{

var id = std.StudentId;

var name = std.StudentName;

var age = std.Age;

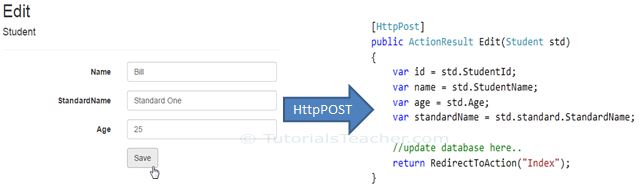
var standardName = std.standard.StandardName;

//update database here..

return RedirectToAction("Index");

}

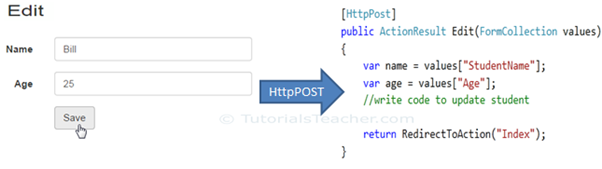
Alors maintenant, framework MVC sera mappe automatiquement les valeurs de collecte de formulaire au paramètre de type étudiant lorsque le formulaire présente une demande http POST pour modifier le mode d'action, comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/model-class-binding.png)modèle de liaison

Ainsi donc, il se lie automatiquement les champs de formulaire au paramètre de type complexe de méthode d'action.

### **FormCollection:**

Vous pouvez également inclure FormCollection paramètre de type dans la méthode au lieu de type complexe d'action, pour récupérer toutes les valeurs de vue des champs de formulaire comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/formcollection.png)modèle de liaison

## **Attribut Bind:**

framework ASP.NET MVC vous permet également de spécifier les propriétés d'une classe de modèle que vous souhaitez lier. Le [Bind] attribut vous permet de spécifier les propriétés exactes d'un liant de modèle doit inclure ou exclure dans la liaison.

Dans l'exemple suivant, Edit méthode d'action ne se lier StudentID et StudentName propriété d'un modèle de l'élève.

exemple Bind:

[HttpPost]

public ActionResult Edit([Bind(Include = "StudentId, StudentName")] Student std)

{

var name = std.StudentName;

//write code to update student

return RedirectToAction("Index");

}

Vous pouvez également utiliser Exclure les propriétés comme ci-dessous.

exemple Bind:

[HttpPost]

public ActionResult Edit([Bind(Exclude = "Age")] Student std)

{

var name = std.StudentName;

//write code to update student

return RedirectToAction("Index");

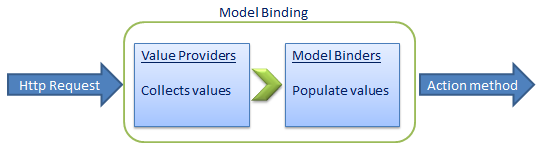
}

L'attribut Bind permettra d'améliorer la performance que de lier les propriétés dont vous aviez besoin.

## **Inside Modèle Reliure:**

Comme vous l'avez vu que la liaison modèle convertit automatiquement les valeurs de demande dans un objet de type primitif ou complexe. Modèle de liaison est un processus en deux étapes. Tout d'abord, il recueille les valeurs de la requête HTTP entrante et d'autre part, remplit type primitif ou d'un type complexe avec ces valeurs.

Les fournisseurs de valeur sont responsables de la collecte des valeurs de demande et du modèle Reliures sont responsables de remplissage de valeurs.

[](http://www.tutorialsteacher.com/Content/images/mvc/model-binding-3.png)Modèle de liaison dans MVC

valeur collection de fournisseur par défaut évalue les valeurs des sources suivantes:

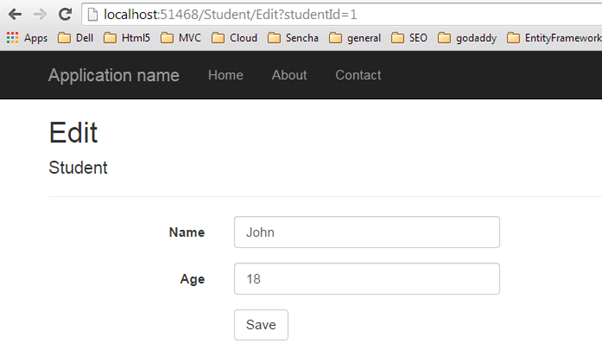
1. Auparavant lié paramètres d'action, lorsque l'action est une action de l'enfant
2. Les champs du formulaire (Request.Form)
3. Les valeurs de propriété dans le JSON Demande corps (Request.InputStream), mais seulement lorsque la demande est une requête AJAX
4. données Route (RouteData.Values)
5. paramètres QueryString (de Request.QueryString)
6. Fichiers Publié (Request.Files)

MVC comprend [DefaultModelBinder](http://msdn.microsoft.com/en-us/library/system.web.mvc.defaultmodelbinder(v=vs.118).aspx) classe qui lie efficacement la plupart des types de modèles.

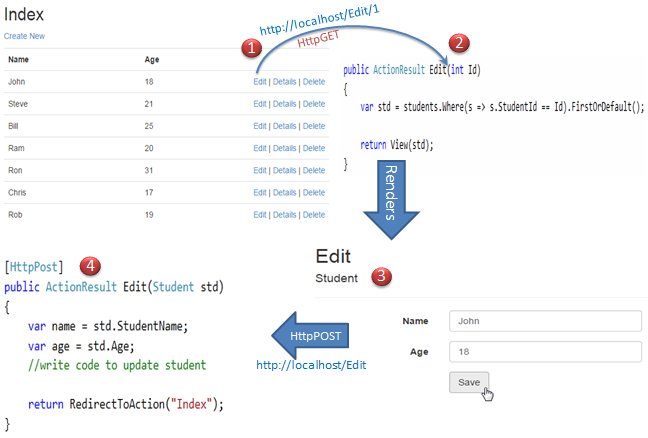
Visitez MSDN pour obtenir des informations détaillées sur la [liaison modèle](http://msdn.microsoft.com/en-us/magazine/hh781022.aspx) .

Créer Modifier Vue:

Nous avons déjà créé la vue Index dans la section précédente. Dans cette section, nous allons créer la vue Éditer en utilisant un modèle d'échafaudage par défaut, comme indiqué ci-dessous.L'utilisateur peut mettre à jour les données des élèves existantes en utilisant la vue Edition.

[](http://www.tutorialsteacher.com/Content/images/mvc/edit-view-4.png)Modifier vue

La vue Edit sera rendu sur le clic sur le bouton Modifier de vue Index. La figure suivante décrit l'ensemble des étapes d'édition.

[](http://www.tutorialsteacher.com/Content/images/mvc/edit-process.png)Étapes d'édition dans MVC

La figure ci-dessus illustre les étapes suivantes.

1. L'utilisateur clique sur le lien Modifier en mode Index qui va envoyer HttpGet demande *http: // localhost / étudiant / modifier / {Id}* avec le paramètre Id correspondant dans la chaîne de requête.Cette demande sera traitée par HttpGet méthode Edit action. (Par la méthode d'action par défaut gère HttpGet demande si aucun attribut spécifié)

2. HttpGet Modifier méthode d'action va chercher les données des élèves à partir de la base de données, en fonction du paramètre Id fourni et rendre la vue Éditer avec ces données d'étudiants particuliers.

3. L'utilisateur peut modifier les données et cliquez sur le bouton Enregistrer dans la vue Edition. Le bouton Enregistrer enverra une demande HttpPost *http: // localhost / Etudiant / Modifier* la collecte de données de formulaire.

4. La méthode d'action HttpPost Modifier dans StudentController va enfin mettre à jour les données dans la base de données et de rendre une page d'index avec les données rafraîchies en utilisant la méthode de RedirectToAction comme une quatrième étape.

Ce sera donc le processus complet afin de modifier les données en utilisant Modifier vue dans ASP.NET MVC.

Commençons donc à mettre en œuvre les étapes ci-dessus.

Nous allons utiliser suivant la classe du modèle de l'élève pour notre Modifier vue.

Modèle Etudiant - C #:

public class Student

{

public int StudentId { get; set; }

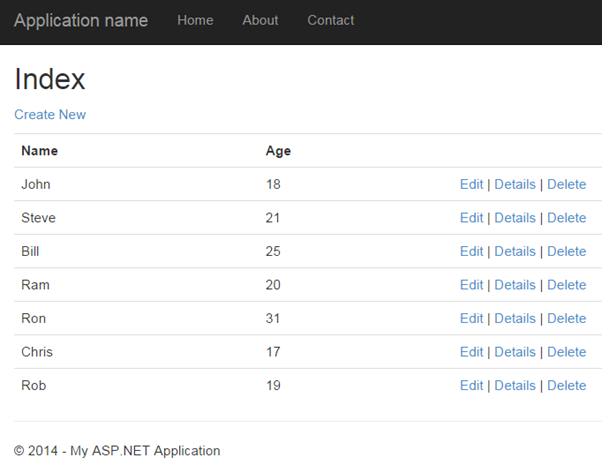
[Display( Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

}

Étape: 1   
Nous avons déjà créé une vue d'index dans la [section précédente](http://www.tutorialsteacher.com/mvc/mvc-view) en utilisant un modèle d'échafaudage de la liste qui comprend un lien Modifier action comme indiqué ci - dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/index-view.png)Index Vue

Un lien Modifier envoie HttpGet demande à la méthode Edit action de StudentController avec StudentID correspondante dans la chaîne de requête. Par exemple, un lien Modifier avec l'étudiant John va ajouter une chaîne StudentID = 1 requête à la demande url car John StudentID est 1. De même tout le lien Modifier comprendra un StudentID respectif dans la chaîne de requête.

Étape 2:   
Maintenant, créer une méthode d'action HttpGet Modifier dans StudentController. La vue de l' indice ci - dessus envoie le paramètre StudentID à la méthode d'action HttpGet Edition sur le clic de lien Modifier.

La méthode d'action HttpGet Edit () doit effectuer deux tâches, elle devrait d'abord chercher l'information des étudiants à partir de la source de données sous-jacente, dont le StudentID correspond à la StudentID dans la chaîne de requête. Deuxièmement, il devrait rendre l'affichage Modifier les informations de l'étudiant afin que l'utilisateur peut mettre à jour.

Ainsi, la méthode d'action Edit () doit avoir un paramètre de StudentID. framework MVC se liera automatiquement une chaîne de requête pour les paramètres d'une méthode d'action si le nom est matchs. faire S'il vous plaît assurer que le nom du paramètre correspond à la chaîne de requête.

Exemple: HttpGet Edit () Méthode d'action - C #

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

IList<Student> studentList = new List<Student>() {

new Student(){ StudentId=1, StudentName="John", Age = 18 },

new Student(){ StudentId=2, StudentName="Steve", Age = 21 },

new Student(){ StudentId=3, StudentName="Bill", Age = 25 },

new Student(){ StudentId=4, StudentName="Ram", Age = 20 },

new Student(){ StudentId=5, StudentName="Ron", Age = 31 },

new Student(){ StudentId=6, StudentName="Chris", Age = 17 },

new Student(){ StudentId=7, StudentName="Rob", Age = 19 }

};

public ActionResult Edit(int StudentId)

{

//Get the student from studentList sample collection for demo purpose.

//You can get the student from the database in the real application

var std = studentList.Where(s => s.StudentId == StudentId).FirstOrDefault();

return View(std);

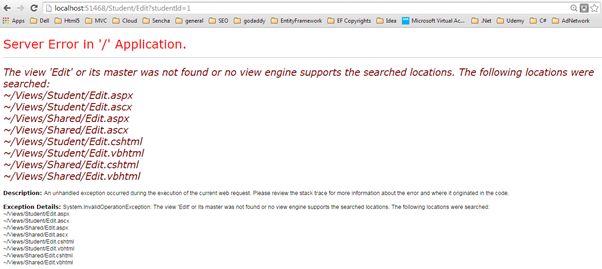
}

}

}

Comme vous pouvez le voir dans la méthode Edit ci-dessus, nous avons utilisé une requête LINQ pour obtenir l'étudiant de la collection studentList échantillon dont StudentID correspond avec StudentID fourni, puis nous injecter cet objet étudiant en View. Dans une application réelle de la vie, vous pouvez obtenir l'étudiant à partir de la base de données au lieu de prélèvement de l'échantillon.

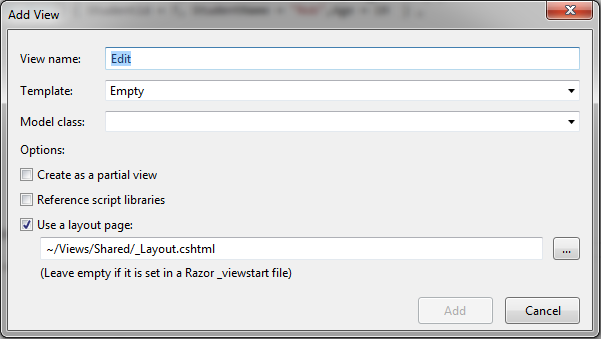
Maintenant, si vous cliquez sur le lien Modifier de la vue Index alors vous obtiendrez l'erreur suivante.

[](http://www.tutorialsteacher.com/Content/images/mvc/edit-view-1.png)Éditer erreur

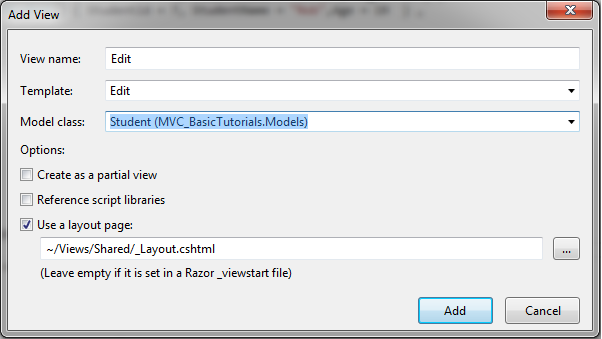
L'erreur ci-dessus a eu lieu parce que nous avons pas encore créé une vue Edition. Par défaut, framework MVC recherchera Edit.cshtml ou Edit.vbhtml ou Edit.aspx ou fichier Edipt.ascx dans View -> Étudiant ou dossier partagé.

Étape 3:   
Pour créer l' affichage Modifier, cliquez à l' intérieur Modifier la méthode d'action et cliquez sur**Ajouter une vue ..** Il ouvrira Ajouter Voir le dialogue.

Dans la zone Ajouter Voir le dialogue, garder le nom de la vue comme Edit. (Vous pouvez modifier selon vos besoins.)

[](http://www.tutorialsteacher.com/Content/images/mvc/edit-view-2.png)Modifier vue

Sélectionnez Modifier dans la liste déroulante Modèle et sélectionnez également des étudiants pour la classe de modèle, comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/edit-view-3.png)Modifier vue

Maintenant, cliquez sur Ajouter pour générer Edit.cshtml vue sous le dossier View / Student comme indiqué ci-dessous.

Edit.cshtml:

@model MVC\_BasicTutorials.Models.Student

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Student</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

@Html.HiddenFor(model => model.StudentId)

<div class="form-group">

@Html.LabelFor(model => model.StudentName, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.StudentName, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.StudentName, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Age, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.Age, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Age, "", new { @class = "text-danger"< })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default" />

</div>

</div>

</div>

}

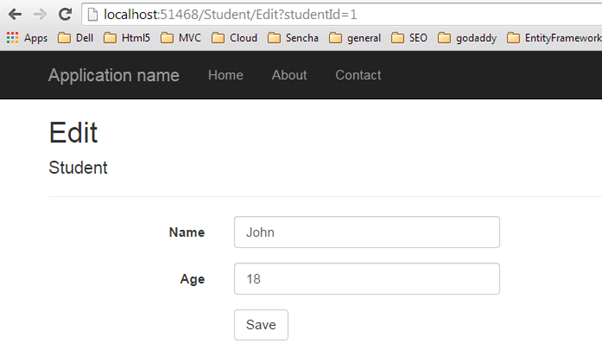
<div>

@Html.ActionLink("Back to List", "Index")

</div>

S'il vous plaît noter que Edit.cshtml comprend HtmlHelper méthode @using (Html.BeginForm())pour créer un élément de formulaire HTML. Html.BeginForm envoie une demande de HttpPost par défaut.

Maintenant, cliquez sur le lien Modifier d'un étudiant dans la vue Index. Modifier vue sera l'information des étudiants d'affichage dont le lien Modifier cliqué, comme illustré ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/edit-view-4.png)Modifier vue

Vous pouvez modifier le nom ou l'âge de l'élève et cliquez sur Enregistrer. Enregistrer méthode devrait envoyer une demande de HttpPost parce que la demande POST envoie les données de formulaire comme une partie de la demande, et non pas dans le querystring. Donc, écrire une méthode POST comme quatrième étape.

Étape 4:   
Maintenant, écrivez POST Modifier la méthode d'action pour sauver l'étudiant édité comme illustré ci - dessous.

C #:

[HttpPost]

public ActionResult Edit(Student std)

{

//write code to update student

return RedirectToAction("Index");

}

Comme vous pouvez le voir dans le code ci - dessus, la méthode Edit () nécessite un objet de l'élève en tant que paramètre d'entrée. La vue Edit () va se lie automatiquement la collecte des données de formulaire pour le paramètre du modèle de l' élève. S'il vous plaît visitez [modèle Reliure](http://www.tutorialsteacher.com/mvc/model-binding-in-asp.net-mvc)section pour plus d' informations. Ici, vous pouvez mettre à jour les informations à la base dedonnées et la rediriger vers Index action. (nous avons pas le code pour mettre à jour la base dedonnées ici pour fins de démonstration écrite)

Maintenant, en cliquant sur le bouton Enregistrer dans la vue Edit enregistrer les informations mises à jour et la rediriger vers la méthode d'action Index ().

De cette façon, vous pouvez fournir des fonctionnalités d'édition en utilisant un défaut échafaudage Modifier modèle. Cependant, vous pouvez également créer une vue Éditer sans utiliser un modèle Modifier échafaudage.

L'exemple suivant montre la classe StudentController avec toutes les méthodes d'action.

Exemple: StudentController - C #

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

IList<Student> studentList = new List<Student>() {

new Student(){ StudentId=1, StudentName="John", Age = 18 },

new Student(){ StudentId=2, StudentName="Steve", Age = 21 },

new Student(){ StudentId=3, StudentName="Bill", Age = 25 },

new Student(){ StudentId=4, StudentName="Ram", Age = 20 },

new Student(){ StudentId=5, StudentName="Ron", Age = 31 },

new Student(){ StudentId=6, StudentName="Chris", Age = 17 },

new Student(){ StudentId=7, StudentName="Rob", Age = 19 }

};

// GET: Student

public ActionResult Index()

{

return View(studentList);

}

public ActionResult Edit(int StudentId)

{

//Get the student from studentList sample collection for demo purpose.

//You can get the student from the database in the real application

var std = studentList.Where(s => s.StudentId == StudentId).FirstOrDefault();

return View(std);

}

[HttpPost]

public ActionResult Edit(Student std)

{

//write code to update student

return RedirectToAction("Index");

}

}

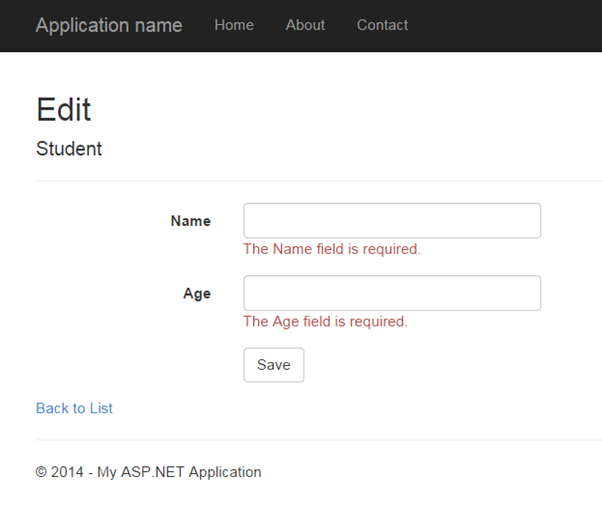
}

Apprenez à mettre en œuvre des validations dans la vue de rasoir dans la section suivante.

# Mettre en œuvre la validation des données dans MVC:

Dans cette section, vous apprendrez comment implémenter des validations de données dans l'application ASP.NET MVC.

Nous avons créé une vue Edition pour étudiants dans la section précédente. Maintenant, nous allons mettre en œuvre la validation des données dans la vue Edition, qui permet d'afficher des messages de validation sur le clic de bouton Enregistrer, comme indiqué ci-dessous si Nom de l'étudiant ou l'âge est vide.

[](http://www.tutorialsteacher.com/Content/images/mvc/validation.png)Validation

## **DataAnnotations:**

ASP.NET MVC utilise DataAnnotations attributs pour mettre en œuvre les validations.DataAnnotations comprend la validation intégrée des attributs pour les différentes règles de validation, qui peuvent être appliquées aux propriétés de la classe de modèle. framework ASP.NET MVC automatiquement appliquer ces règles de validation et les messages de validation d'affichage dans la vue.

Les attributs de DataAnnotations inclus dans *System.ComponentModel.DataAnnotations* espace denoms. Le tableau suivant la validation des listes de DataAnnotations attributs.

| **Attribut** | **La description** |
| --- | --- |
| Obligatoire | Indique que la propriété est un champ obligatoire |
| StringLength | Définit une longueur maximale pour le champ de chaîne |
| Gamme | Définit une valeur maximale et minimale pour un champ numérique |
| Expression régulière | Indique que la valeur du champ doit correspondre spécifié Expression régulière |
| Carte de crédit | Indique que le champ spécifié est un numéro de carte de crédit |
| CustomValidation | Spécifié méthode de validation personnalisé pour valider le champ |
| Adresse e-mail | Validation avec le format d'adresse e-mail |
| Extension de fichier | Validation avec l'extension de fichier |
| Longueur maximale | Indique la longueur maximale pour un champ de chaîne |
| MinLength | Indique la longueur minimale pour un champ de chaîne |
| Téléphone | Indique que le champ est un numéro de téléphone en utilisant l'expression régulière pour les numéros de téléphone |

Commençons à mettre en œuvre la validation dans l'affichage Modifier pour les étudiants.

**Étape 1:** Tout d' abord, appliquer l' attribut DataAnnotation sur les propriétés de l' élève modèle declasse. Nous voulons valider que StudentName et l' âge est pas vide. En outre, l' âge doit se situer entre 5 et 50. Visitez [Modèle](http://www.tutorialsteacher.com/mvc/mvc-model) section si vous ne savez pas comment créer une classe de modèle.

Exemple: Appliquer des attributs de DataAnnotation

public class Student

{

public int StudentId { get; set; }

[Required]

public string StudentName { get; set; }

[Range(5,50)]

public int Age { get; set; }

}

http://www.tutorialsteacher.com/Content/images/tips.pngVous pouvez également appliquer plusieurs DataAnnotations attributs de validation pour une seule propriété si nécessaire.

Dans l'exemple ci - dessus, nous avons appliqué un ***Required***attribut à la propriété StudentName. Alors maintenant, le framework MVC affiche automatiquement le message d'erreur par défaut, si l'utilisateur tente d'enregistrer le formulaire demodification sans entrer le nom de l' étudiant. De la même manière, le ***Range*** attribut est appliqué avec une valeur minimum et maximum à la propriété Age. Cela permettra de valider et d' afficher un message d'erreur si l'utilisateur a soit pas entré Âge ou entré un âge inférieur à 5 ou plus de 50.

**Etape 2:** Créer l'EEG et la méthode POST Modifier l' action dans la même section précédente. La méthode d'action GET rendra Modifier afin de modifier l'étudiant sélectionné et la méthode POST Modifier sauvera étudiant édité comme illustré ci - dessous.

Exemple: Modifier les modes d'action:

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

public ActionResult Edit(int id)

{

var std = studentList.Where(s => s.StudentId == StudentId)

.FirstOrDefault();

return View(std);

}

[HttpPost]

public ActionResult Edit(Student std)

{

if (ModelState.IsValid) {

//write code to update student

return RedirectToAction("Index");

}

return View(std);

}

}

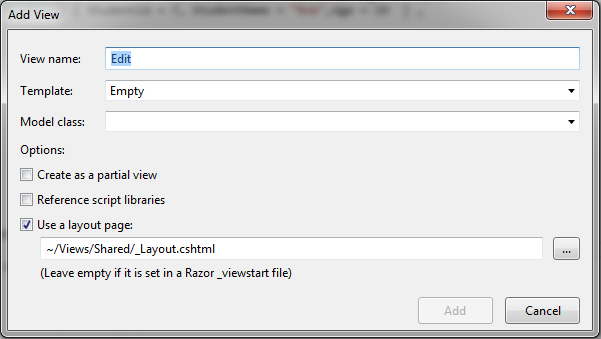
}

Comme vous pouvez le voir dans la méthode POST Edition, nous vérifions d'abord si le ModelState est valide ou non. Si ModelState est valide puis mettre à jour l'étudiant dans la base de données, sinon, retourner l'affichage Modifier à nouveau avec les mêmes données sur les élèves.

ModelState.IsValid détermine que les valeurs que soumises satisfont tous les attributs de validation de DataAnnotation appliqués pour modéliser les propriétés.

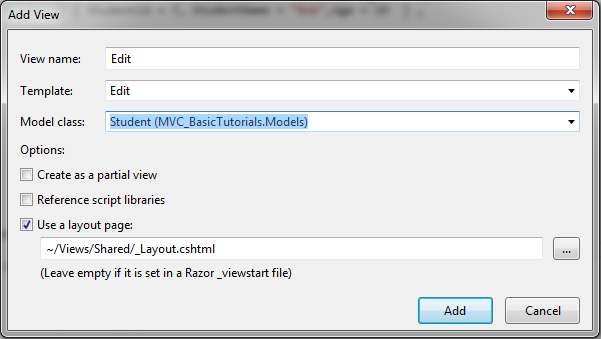
**Étape 3:** Maintenant, créez une vue Edition pour étudiants.

Pour créer une vue Edition, cliquez sur Modifier à l' intérieur méthode d'action -> cliquez sur **Ajouter une vue ..**

[](http://www.tutorialsteacher.com/Content/images/mvc/addview.png)Créer Éditer

Dans la zone Ajouter Voir le dialogue, garder le nom de la vue comme Edit. (Vous pouvez modifier selon vos besoins.)

Sélectionnez le modèle Modifier dans la liste déroulante Modèle et sélectionnez également étudiant de classe modèle, comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/addview-2.png)Créer Éditer

Maintenant, cliquez sur **Ajouter** pour générer Modifier vue sous le dossier View / étudiant.Edit.cshtml sera généré comme indiqué ci - dessous.

Edit.cshtml:

@model MVC\_BasicTutorials.Models.Student

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Student</h4>

<hr />

**@Html.ValidationSummary(true, "", new { @class = "text-danger" })**

@Html.HiddenFor(model => model.StudentId)

<div class="form-group">

@Html.LabelFor(model => model.StudentName, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.StudentName, new { htmlAttributes = new { @class = "form-control" } })

**@Html.ValidationMessageFor(model => model.StudentName, "", new { @class = "text-danger" })**

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Age, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.Age, new { htmlAttributes = new { @class = "form-control" } })

**@Html.ValidationMessageFor(model => model.Age, "", new { @class = "text-danger" })**

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default" />

</div>

</div>

</div>

}

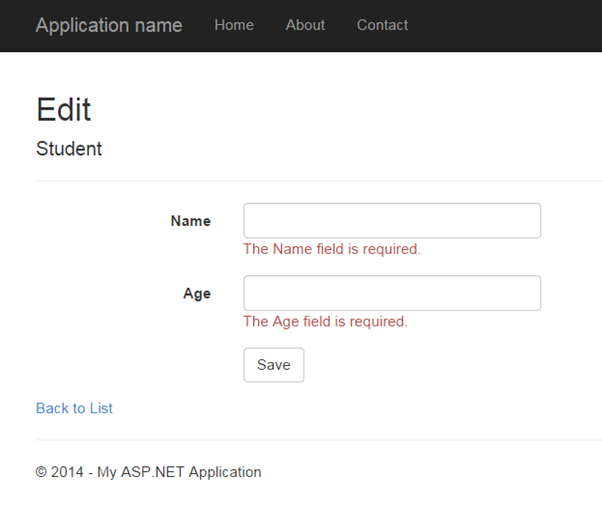
<div>

@Html.ActionLink("Back to List", "Index")

</div>

Comme vous pouvez le voir dans ce qui précède Edit.cshtml, il appelle Html méthode Helper**ValidationMessageFor** pour chaque champ et **ValidationSummary** méthode au sommet.ValidationMessageFor est responsable d'afficher un message d'erreur pour le champ spécifié.ValidationSummary affiche une liste de tous les messages d'erreur à la fois.

Alors maintenant, il affiche un message de validation par défaut lorsque vous soumettez un formulaire de modification sans entrer un nom ou l'âge.

[](http://www.tutorialsteacher.com/Content/images/mvc/validation-editview.png)

Validation

Ainsi, vous pouvez mettre en œuvre des validations en appliquant divers attributs de DataAnnotation à la classe de modèle et en utilisant ValidationMessage () ou la méthode ValidationMessageFor () dans la vue.

### **Lectures complémentaires:**

* [Mettre en œuvre client validation côté dans MVC View.](http://www.tutorialsteacher.com/articles/enable-client-side-valiation-in-mvc)



### **Points à retenir:**

1. ASP.NET MVC utilise DataAnnotations attributs pour validation.
2. DataAnnotations attributs peuvent être appliqués sur les propriétés de la classe de modèle pour indiquer le type de valeur de la propriété tiendra.
3. La validation attributs suivants disponibles par défaut
   1. Obligatoire
   2. StringLength
   3. Gamme
   4. Expression régulière
   5. Carte de crédit
   6. CustomValidation
   7. Adresse e-mail
   8. Extension de fichier
   9. Longueur maximale
   10. MinLength
   11. Téléphone
4. Utilisez **ValidationSummary** pour afficher tous les messages d'erreur dans la vue.
5. Utilisez **ValidationMessageFor** ou **ValidationMessage** méthode d'aide pour afficher des messages d'erreur au niveau du terrain dans la vue.
6. Vérifiez si le modèle est valide avant la mise à jour dans la méthode d'action en utilisant ModelState.IsValid.
7. Activer la validation du côté client pour afficher des messages d'erreur sans effet de postback dans le navigateur.

# ValidationMessage:

Vous avez appris à mettre en œuvre la validation dans une vue dans la section presious. Ici, nous allons voir la méthode extension HtmlHelper ValidtionMessage en détail.

Le Html.ValidationMessage () est une méthode d'extension, qui est une méthode faiblement typé. Il affiche un message de validation si une erreur existe pour le champ spécifié dans l'objet ModelStateDictionary.

## **ValidationMessage () Signature:**

MvcHtmlString ValidateMessage(string modelName, string validationMessage, object htmlAttributes)

Visitez MSDN de connaître toutes les [surcharges de méthode ValidationMessage ()](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.validationextensions.validationmessage(v=vs.118).aspx) .

Prenons l'exemple ValidationMessage suivant.

Exemple: ValidationMessage

@model Student

@Html.Editor("StudentName") <br />

@Html.ValidationMessage("StudentName", "", new { @class = "text-danger" })

Dans l'exemple ci-dessus, le premier paramètre dans la méthode ValidationMessage est un nom de propriété pour laquelle nous voulons montrer le message d'erreur, par exemple StudentName. Le deuxième paramètre est pour le message d'erreur personnalisé et le troisième paramètre est pour les attributs HTML comme css, le style, etc.

La méthode ValidationMessage () n'affichera une erreur, si vous avez configuré les DataAnnotations attribuent à la propriété spécifiées dans la classe de modèle. Ce qui suit est un modèle de classe d'étudiant où les DataAnnotations attribut "Obligatoire" est appliqué à la propriété StudentName.

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Required]

public string StudentName { get; set; }

public int Age { get; set; }

}

Le code ci-dessus va générer suivant html.

Résultat Html:

<input id="StudentName"

name="StudentName"

type="text"

value="" />

<span class="field-validation-valid text-danger"

data-valmsg-for="StudentName"

data-valmsg-replace="true">

</span>

Maintenant, lorsque l'utilisateur soumet un formulaire sans entrer dans un StudentName, puis ASP.NET MVC utilise un attribut de données de Html5 pour la validation et un message de validation par défaut sera injecté, lorsque l'erreur de validation se produit, comme indiqué ci-dessous.

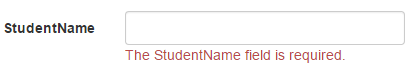
Html avec le message de validation:

<span class="field-validation-error text-danger"

data-valmsg-for="StudentName"

data-valmsg-replace="true">The StudentName field is required.</span>

Le message d'erreur ressemblera ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/validationmessage.png)Sortie de ValidationMessage () Méthode

## **Message d'erreur personnalisé:**

Vous pouvez afficher votre propre message d'erreur au lieu du message d'erreur par défaut, comme indiqué ci-dessus. Vous pouvez fournir un message d'erreur personnalisé soit dans les DataAnnotations attribut ou une méthode ValidationMessage ().

Utilisez le paramètre de l'DataAnnotation attribue à fournir votre propre message d'erreur personnalisé, comme indiqué ci-dessous.

message d'erreur personnalisé dans le modèle Exemple:

public class Student

{

public int StudentId { get; set; }

[Required(ErrorMessage="Please enter student name.")]

public string StudentName { get; set; }

public int Age { get; set; }

}

En outre, vous pouvez spécifier un message en tant que deuxième paramètre dans la méthode ValidationMessage (), comme illustré ci-dessous.

Exemple: message d'erreur personnalisée

@model Student

@Html.Editor("StudentName") <br />

@Html.ValidationMessage("StudentName", "Please enter student name.", new { @class = "text-danger" })

En savoir plus sur la méthode ValidationMessageFor dans la section suivante.

# ValidationMessageFor:

Le Html.ValidationMessageFor () est une méthode d'extension fortement typé. Il affiche un message de validation si une erreur existe pour le champ spécifié dans l'objet ModelStateDictionary.

## **ValidationMessageFor () Signature:**

MvcHtmlString ValidateMessage(Expression<Func<dynamic,TProperty>> expression, string validationMessage, object htmlAttributes)

Visitez MSDN de connaître toutes les [surcharges de méthode ValidationMessageFor ()](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.validationextensions.validationmessagefor(v=vs.118).aspx) .

Prenons l'exemple suivant ValidationMessageFor ().

Exemple: ValidationMessageFor

@model Student

@Html.EditorFor(m => m.StudentName) <br />

@Html.ValidationMessageFor(m => m.StudentName, "", new { @class = "text-danger" })

Dans l'exemple ci-dessus, le premier paramètre dans la méthode ValidationMessageFor est une expression lambda pour définir une propriété pour laquelle nous voulons montrer le message d'erreur.Le deuxième paramètre est pour le message d'erreur personnalisé et le troisième paramètre est pour les attributs HTML comme css, le style, etc.

La méthode ValidationMessageFor () n'affichera une erreur si vous avez configuré DataAnnotations attribuent à la propriété spécifiées dans la classe de modèle. L'exemple suivant est une classe de modèle d'étudiant où les DataAnnotations attribut "Obligatoire" est appliqué à la propriété StudentName.

Exemple: Modèle Etudiant

public class Student

{

public int StudentId { get; set; }

[Required]

public string StudentName { get; set; }

public int Age { get; set; }

}

Le code ci-dessus va générer le code HTML suivant.

Résultat Html:

<input id="StudentName"

name="StudentName"

type="text"

value="" />

<span class="field-validation-valid text-danger"

data-valmsg-for="StudentName"

data-valmsg-replace="true">

</span>

Maintenant, lorsque l'utilisateur soumet un formulaire sans entrer dans le StudentName puis ASP.NET MVC utilise l'attribut de données de Html5 pour la validation et le message de validation par défaut sera injecté lorsqu'une erreur de validation se produit, comme indiqué ci-dessous.

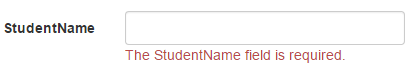
Html avec le message de validation:

<span class="field-validation-error text-danger"

data-valmsg-for="StudentName"

data-valmsg-replace="true">The StudentName field is required.</span>

Le message d'erreur apparaîtra comme l'image ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/validationmessage.png)Sortie de ValidationMessageFor () Méthode

## **Message d'erreur personnalisé:**

Vous pouvez afficher votre propre message d'erreur au lieu du message d'erreur par défaut comme ci-dessus. Vous pouvez fournir un message d'erreur personnalisé soit dans l'attribut DataAnnotations ou la méthode ValidationMessageFor ().

Utilisez le paramètre ErrorMessage de l'DataAnnotation attribue à fournir votre propre message d'erreur personnalisé, comme indiqué ci-dessous.

message d'erreur personnalisé dans le modèle Exemple:

public class Student

{

public int StudentId { get; set; }

[Required(ErrorMessage="Please enter student name.")]

public string StudentName { get; set; }

public int Age { get; set; }

}

En outre, vous pouvez spécifier un message en tant que deuxième paramètre dans la méthode ValidationMessage (), comme illustré ci-dessous.

Exemple: message d'erreur personnalisée

@model Student

@Html.Editor("StudentName") <br />

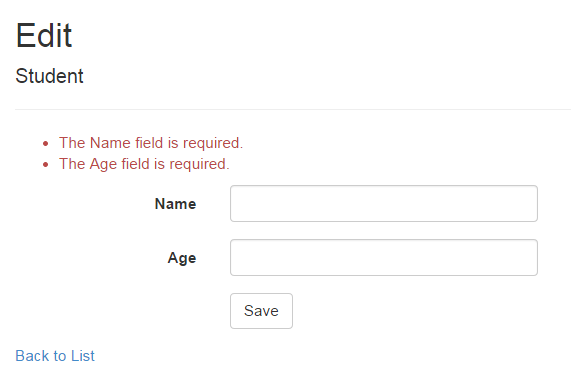
@Html.ValidationMessageFor(m => m.StudentName, "Please enter student name.", new { @class = "text-danger" })

En savoir plus sur la méthode ValidationSummary dans la section suivante.

# ValidationSummary:

La méthode d'assistance ValidationSummary génère une liste non ordonnée (élément ul) des messages de validation qui sont l'objet ModelStateDictionary.

Le ValidationSummary peut être utilisé pour afficher tous les messages d'erreur pour tous les domaines. Il peut également être utilisé pour afficher des messages d'erreur personnalisés. La figure suivante montre comment ValidationSummary affiche les messages d'erreur.

[](http://www.tutorialsteacher.com/Content/images/mvc/validationsummary.png)ValidationSummary

## **ValidationSummary () Signature:**

MvcHtmlString ValidateMessage(bool excludePropertyErrors, string message, object htmlAttributes)

Visitez MSDN de connaître toutes les [surcharges de méthode ValidationMessage ()](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.validationextensions.validationsummary(v=vs.118).aspx) .

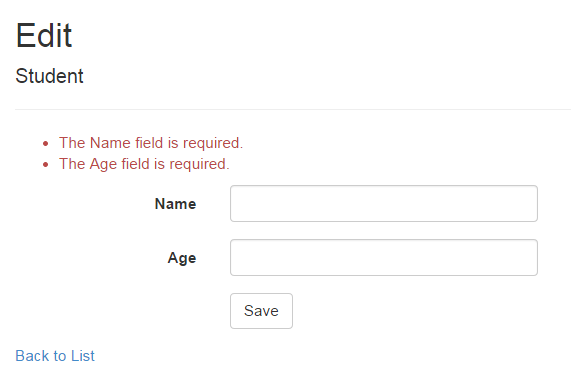
## **les messages d'erreur au niveau du champ d'affichage en utilisant ValidationSummary:**

Par défaut, ValidationSummary filtre les messages d'erreur au niveau du terrain. Si vous souhaitez afficher des messages d'erreur au niveau du terrain comme un résumé puis spécifiez excludePropertyErrors = false.

Exemple: ValidationSummary pour afficher les erreurs de champ

@Html.ValidationSummary(false, "", new { @class = "text-danger" })

Alors maintenant, la vue Éditer suivant affiche des messages d'erreur comme un résumé en haut.faire S'il vous plaît assurer que vous ne disposez pas d'une méthode ValidationMessageFor pour chacun des champs.

[](http://www.tutorialsteacher.com/Content/images/articles/validationsummary-demo.png)message d'erreur d'affichage en utilisant ValidationSymmary

## **Afficher les messages d'erreur personnalisés:**

Vous pouvez également afficher un message d'erreur personnalisé en utilisant ValidationSummary.Par exemple, nous voulons afficher un message si Nom de l'élève existe déjà dans la base de données.

Pour afficher un message d'erreur personnalisé, tout d'abord, vous devez ajouter les erreurs personnalisées dans le ModelState dans la méthode d'action approprié.

Exemple: Ajouter erreur dans ModelState

if (ModelState.IsValid) {

//check whether name is already exists in the database or not

bool nameAlreadyExists = \* check database \*

if(nameAlreadyExists)

{

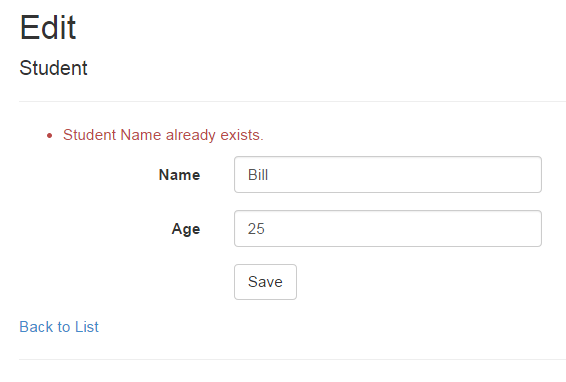
ModelState.AddModelError(string.Empty, "Student Name already exists.");

return View(std);

}

}

Comme vous pouvez le voir dans le code ci-dessus, nous avons ajouté des messages d'erreur personnalisés à l'aide de la méthode ModelState.AddModelError. La méthode ValidationSummary affichera automatiquement tous les messages ajoutés dans ModelState d'erreur.

[](http://www.tutorialsteacher.com/Content/images/articles/validationsummary-demo2.png)message d'erreur d'affichage en utilisant ValidationSymmary

Ainsi, vous pouvez utiliser la méthode d'assistance ValidationSummary pour afficher les messages d'erreur.

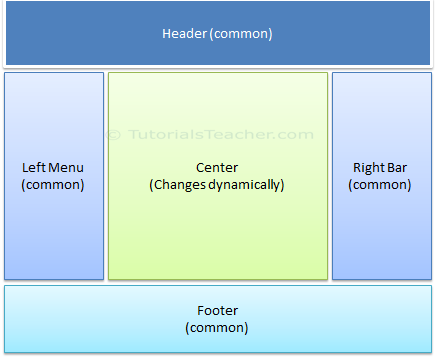
# Layout View:

Dans cette section, vous apprendrez à connaître le point de vue de la mise en page dans ASP.NET MVC.

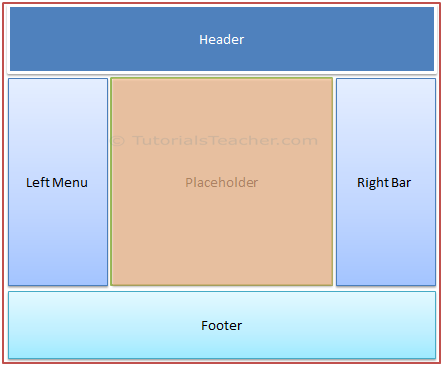
## **Quelle est la position de la mise en page?**

Une application peut contenir des parties communes dans l'interface utilisateur, qui reste la même dans toute l'application, comme le logo, en-tête, la barre de navigation de gauche, barre de droite ou de la section de pied de page. ASP.NET MVC introduit une vue de la mise en page qui contient ces parties de l'interface utilisateur communs, de sorte que nous ne disposons pas d'écrire le même code dans chaque page. La vue de la mise en page est la même que la page principale de l'application Webform ASP.NET.

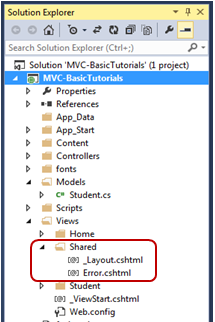
Par exemple, une interface utilisateur de l'application peut contenir en-tête, la barre de menu de gauche, barre de droite et de section de pied qui reste même à chaque page et seule la section centrale modifie de manière dynamique, comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/layout-view-1.png)Exemples de l'interface utilisateur de l'application Pièces

La vue de la mise en page vous permet de définir un modèle de site commun, qui peut être hérité dans plusieurs vues pour fournir un aspect cohérent dans plusieurs pages d'une application. La vue de la mise en page élimine codage double et améliore la vitesse de développement et d'entretien facile.La vue de la mise en page pour l'interface utilisateur de l'échantillon ci-dessus devrait contenir un en-tête, Menu Gauche, Droite et sections de pied de page. Il contient un espace réservé pour la partie centrale qui change de façon dynamique, comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/layout-view-2.png)Layout View

La vue rasoir de mise en page a même extension que d' autres points de vue, .cshtml ou .vbhtml.Vues de mise en page sont partagés avec de multiples points de vue, il doit être stocké dans le dossier partagé. Par exemple, lorsque nous avons créé notre [première application MVC](http://www.tutorialsteacher.com/mvc/create-first-asp.net-mvc-application) dans la section précédente, il a également créé \_Layout.cshtml dans le dossier partagé comme indiqué ci -dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/layout-view-3.png)Voir la mise en page dans le dossier partagé

Ce qui suit est un \_Layout.cshtml généré automatiquement.

\_Layout.cshtml:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title - My ASP.NET Application</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("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul>

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

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

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

Comme vous pouvez le voir, la vue de la mise en page contient html Doctype, tête et corps comme html normal, la seule différence est appel à RenderBody () et RenderSection () méthodes.RenderBody agit comme un espace réservé pour d'autres points de vue. Par exemple, Index.cshtml dans le dossier de la maison sera injecté et rendu dans la vue de la mise en page, où la méthode RenderBody () est appelée. Vous apprendrez au sujet de ces méthodes de rendu plus loin dans cette section.

## **Utilisez Layout View:**

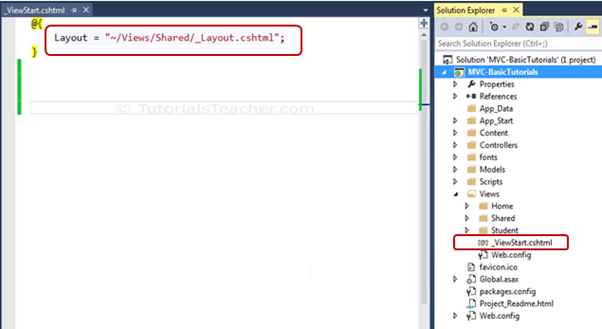
Vous devez vous demander que comment le View savoir quelle disposition de vue à utiliser?

Vous pouvez définir le mode d'affichage de multiples façons, en utilisant \_ViewStart.cshtml ou la mise en place chemin de la mise en page en utilisant la propriété mise en page dans la vue individuelle ou en spécifiant la mise en page nom de la vue dans la méthode d'action.

### **\_ViewStart.cshtml:**

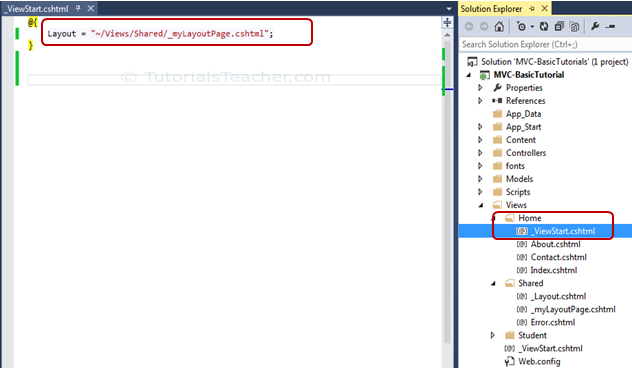
\_ViewStart.cshtml Est inclus dans le dossier Vues par défaut. Elle met en place la mise en page par défaut pour tous les points de vue dans le dossier et ses sous-dossiers en utilisant la propriété mise en page. Vous pouvez attribuer un chemin valide d'une page de mise en page à la propriété de la mise en page.

Par exemple, le \_ViewStart.cshtml suivant dans le **Vues** dossier, définit la propriété Mise à "~ / Views / Shared / \_Layout.cshtml". Alors maintenant, \_layout.cshtml serait vue mise en page de tous les points de vue inclus dans les vues et ses sous - dossiers. Donc , par défaut, toutes les vues dérivées de mise en page par défaut de \_ViewStart.cshtml du dossier Vues.

[](http://www.tutorialsteacher.com/Content/images/mvc/viewstart.png)\_ViewStart.cshtml

\_ViewStart.cshtml Peut également être inclus dans le dossier sous Vue dossier pour définir la mise en page par défaut pour tous les points de vue inclus dans ce sous-dossier particulier seulement.

Par exemple, le \_ViewStart.cshtml suivant dans le dossier Accueil, définit la propriété de mise en page pour \_myLayoutPage.cshtml. Donc, ce \_ViewStart.cshtml influencera toutes les vues incluses dans le dossier d'accueil uniquement. Alors maintenant, Index, A propos et contact vues sera rendu dans \_myLayoutPage.cshtml au lieu de défaut \_Layout.cshml.

[](http://www.tutorialsteacher.com/Content/images/mvc/viewstart-home.png)Layout View

### **Réglage de la propriété de la mise en page en vue individuel:**

Vous pouvez également remplacer de mise en page par défaut défini par \_ViewStart.cshtml en définissant la propriété mise en page dans chaque vue individuel .cshtml. Par exemple, la vue suivante Index utiliser \_myLayoutPage.cshtml même si \_ViewStart.cshtml réglé \_Layout.cshtml.

Indice de vue:

@{

ViewBag.Title = "Home Page";

Layout = "~/Views/Shared/\_myLayoutPage.cshtml";

}

<div class="jumbotron">

<h1>ASP.NET</h1>

<p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.</p>

<p><a href="http://asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>

</div>

<div class="row">

<div class="col-md-4">

<h2>Getting started</h2>

<p>

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that

enables a clean separation of concerns and gives you full control over markup

for enjoyable, agile development.

</p>

<p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301865">Learn more &raquo;</a></p>

</div>

<div class="col-md-4">

<h2>Get more libraries</h2>

<p>NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.</p>

<p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301866">Learn more &raquo;</a></p>

</div>

<div class="col-md-4">

<h2>Web Hosting</h2>

<p>You can easily find a web hosting company that offers the right mix of features and price for your applications.</p>

<p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301867">Learn more &raquo;</a></p>

</div>

</div>

### **Spécifiez la mise en page dans la méthode ActionResult:**

Vous pouvez également spécifier la mise en page à utiliser dans tout en rendant vue méthode d'action en utilisant View () méthode.

L'exemple suivant, la méthode View () rend Vue d'index en utilisant \_myLayoutPage.cshtml.

C #:

public class HomeController : Controller

{

public ActionResult Index()

{

return View("Index", "\_myLayoutPage");

}

public ActionResult About()

{

return View();

}

public ActionResult Contact()

{

return View();

}

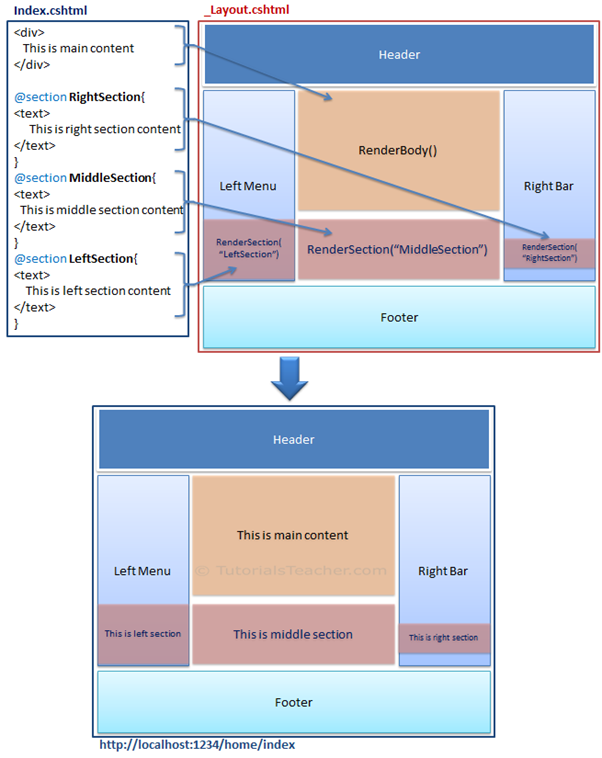
}

## **Méthodes de rendu:**

ASP.NET MVC vue mise en page rend vues enfants en utilisant les méthodes suivantes.

| **méthode** | **La description** |
| --- | --- |
| RenderBody () | Rend la partie de la vue des enfants qui ne sont pas dans une section nommée.vue Mise en page doit inclure la méthode RenderBody (). |
| RenderSection (string name) | Rend une teneur en section nommée et indique si la section est nécessaire.RenderSection () est facultative dans la vue de la mise en page. |

La figure suivante illustre l'utilisation de méthodes RenderBody et RenderSection.

[](http://www.tutorialsteacher.com/Content/images/mvc/layout-view-rendering.png)rendu Méthodes

Comme vous pouvez le voir dans la figure ci-dessus, \_Layout.cshtml comprend la méthode de RenderBody () et la méthode RenderSection (). méthodes de RenderSection spécifient le nom d'une section telle que LeftSection, MiddleSection et RightSection dans la figure ci-dessus. Index.cshtml définit la section nommée en utilisant @section où le nom de chaque section correspond au nom spécifié dans la méthode de RenderSection de \_Layout.cshtml, comme @section RightSection etc. Au moment de l'exécution, les sections nommées de Index.cshtml tels que LeftSection, RightSection et MiddleSection sera rendu à l'endroit approprié où la méthode de RenderSection est appelée. Reste de la partie de vue Index, qui n’est en aucun de la section nommée rendra où la méthode RenderBody () est appelée.

Nous allons créer une nouvelle vue de la mise en page pour comprendre ce qui précède méthodes dans la section Render suivant.

### **Lectures complémentaires :**

* [Différence entre les méthodes de RenderBody et RenderSection](http://www.tutorialsteacher.com/articles/difference-between-renderbody-and-rendersection-mvc)



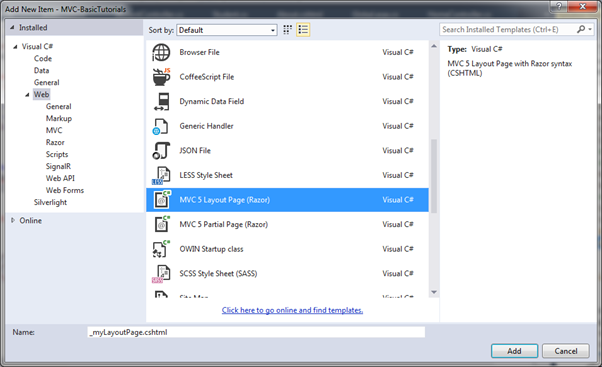
### **Points à retenir:**

1. La vue de la mise en page contient des parties communes d'une interface utilisateur. Il est même comme masterpage de webforms ASP.NET.
2. \_ViewStart.cshtml Fichier peut être utilisé pour spécifier le chemin de la mise en page, qui à son tour sera applicable à toutes les vues du dossier et ses sous-dossiers.
3. Vous pouvez définir la propriété de la mise en page dans la vue individuelle également, pour remplacer réglage par défaut de mise en page de \_ViewStart.cshtml
4. vue Mise en page utilise deux méthodes de rendu: RenderBody () et RenderSection ().
5. RenderBody peut être utilisé qu'une seule fois dans la vue de la mise en page, alors que la méthode RenderSection peut être appelé de temps multiple avec nom différent.
6. méthode RenderBody rend tout le contenu de vue qui ne sont pas enveloppés dans la section nommée.
7. méthode RenderSection rend le contenu d'une vue qui est enveloppé dans la section nommée.
8. RenderSection peut être configuré selon les besoins ou en option. Si nécessaire, tous les points de vue de l'enfant doivent inclure cette section nommée.

Créer Layout View:

Pour créer une nouvelle vue de la mise en page dans Visual Studio, cliquez droit sur le dossier partagé -> sélectionnez Ajouter -> cliquez sur **Nouvel élément ..**

Dans la nouvelle boîte de dialogue Ajouter de l' élément, sélectionnez MVC 5 Mise en page (Razor) et donner à la mise en page nom de la page comme "\_myLayoutPage.cshtml" et cliquez sur **Ajouter** .

[](http://www.tutorialsteacher.com/Content/images/mvc/create-layout-view-1.png)rendu Méthodes

Vous verrez \_myLayoutPage.cshtml comme indiqué ci-dessous.

\_myLayoutPage.cshtml:

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

</head>

<body>

<div>

@RenderBody()

</div>

</body>

</html>

Maintenant, ajoutez la balise <footer> avec la RenderSection("footer",true)méthode alongwith unpeu de style , comme illustré ci - dessous. S'il vous plaît noter que nous fait cette section au besoin.Cela signifie que tout vue qui utilise \_myLayoutPage comme son point de vue de la mise en page doit inclure une section de pied de page.

\_myLayoutPage.cshtml:

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

<div>

@RenderBody()

</div>

<footer class="panel-footer">

@RenderSection("footer", true)

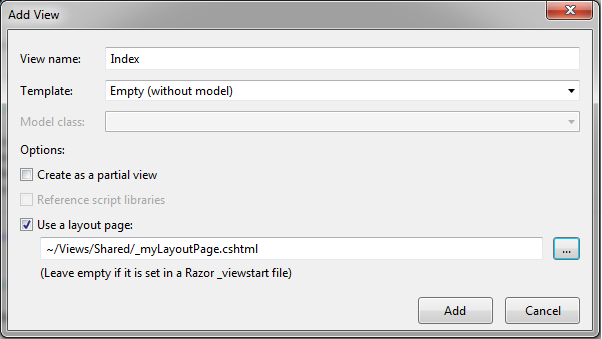
</footer>

</body>

</html>

Maintenant, nous allons utiliser ce \_myLayoutPage.cshtml avec la vue Index de HomeController.

Vous pouvez ajouter un index vue vide par un clic droit sur la méthode de HomeController Index d'action et sélectionnez Ajouter une vue. Sélectionnez Empty comme un modèle d'échafaudage et \_myLayoutPage.cshtml en vue de la mise en page et cliquez sur Ajouter.

[](http://www.tutorialsteacher.com/Content/images/mvc/layout-view-4.png)Ajouter Index Vue

Cela va créer Index.cshtml comme indiqué ci-dessous.

Indice de vue:

@{

ViewBag.Title = "Home Page";

Layout = "~/Views/Shared/\_myLayoutPage.cshtml";

}

<h2>Index</h2>

Alors maintenant, nous avons créé vue Index qui utilise notre \_myLayoutPage.cshtml en vue de la mise en page. Nous allons maintenant ajouter la section de pied de page avec un peu de style parce \_myLayoutPage nécessite section de pied de page.

Indice de vue:

@{

ViewBag.Title = "Home Page";

Layout = "~/Views/Shared/\_myLayoutPage.cshtml";

}

<div class="jumbotron">

<h2>Index</h2>

</div>

<div class="row">

<div class="col-md-4">

<p>This is body.</p>

</div>

@section footer{

<p class="lead">

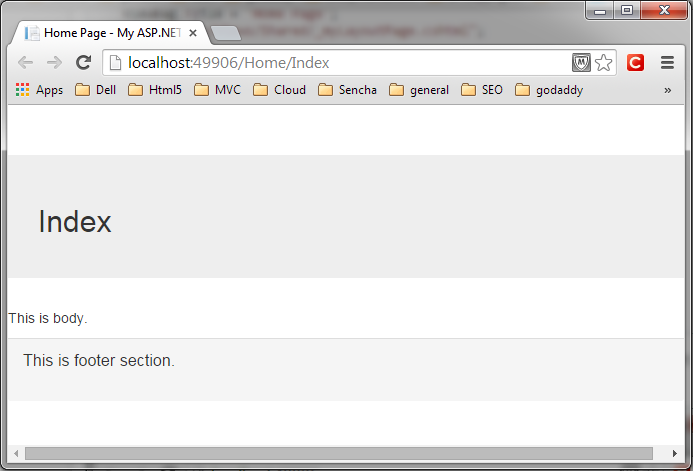
This is footer section.

</p>

}

</div>

Maintenant, exécutez l'application et vous verrez vue Index contiendra corps et pied de page, comme illustré ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/layout-view-5.png)Index Vue

Ainsi, vous pouvez créer un nouveau mode d'affichage avec différentes méthodes de rendu.

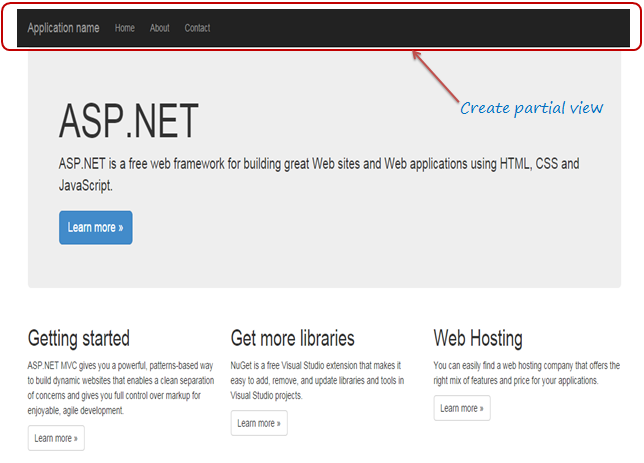
# Vue partielle:

Dans cette section, vous apprendrez au sujet des vues partielles en ASP.NET MVC.

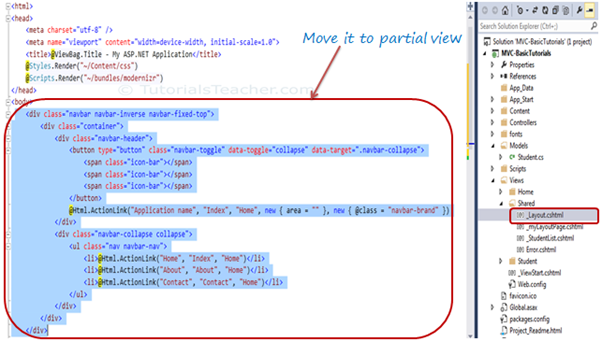
## **Qu'est-ce que partiel View?**

Vue partielle est une vue réutilisable, qui peut être utilisé comme une vue de l'enfant dans plusieurs autres points de vue. Il élimine le codage en double en réutilisant même vue partielle en plusieurs endroits. Vous pouvez utiliser la vue partielle dans la vue de la mise en page, ainsi que d'autres points de vue du contenu.

Pour commencer, nous allons créer une vue partielle simple pour la barre de navigation suivante à des fins de démonstration. Nous allons créer une vue partielle pour elle, de sorte que nous pouvons utiliser la même barre de navigation dans plusieurs vues de mise en page sans avoir à réécrire le même code partout.

[](http://www.tutorialsteacher.com/Content/images/mvc/partial-view-1.png)Vue partielle

La figure suivante montre le code html pour la barre de navigation ci-dessus. Nous allons couper et coller ce code dans une vue partielle individuelle à des fins de démonstration.

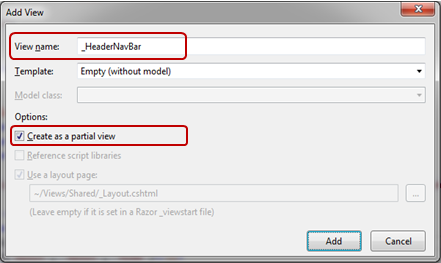
[](http://www.tutorialsteacher.com/Content/images/mvc/partial-view-2.png)Vues partielles

## **Créer un nouveau Vue partielle:**

Pour créer une vue partielle, clic droit sur le dossier partagé -> sélectionnez **Ajouter** -> cliquer sur**View ..**

**Remarque:** Si une vue partielle sera partagée avec plusieurs vues du dossier de contrôleur différent puis créez dans le dossier partagé, sinon vous pouvez créer la vue partielle dans le même dossier où il va être utilisé.

Dans la zone Ajouter Voir le dialogue, entrez Voir nom et sélectionnez "Créer une vue partielle" case à cocher et cliquez sur Ajouter.

[](http://www.tutorialsteacher.com/Content/images/mvc/partial-view-4.png)Vues partielles

On ne va pas utiliser un modèle pour cette vue partielle, donc gardez la liste déroulante Modèle comme vide (sans modèle) et cliquez sur **Ajouter** . Cela va créer une vue partielle vide dans ledossier partagé.

Maintenant, vous pouvez couper le code ci-dessus pour la barre de navigation et de le coller dans \_HeaderNavBar.cshtml comme indiqué ci-dessous:

Vue partielle: \_HeaderNavBar.cshtml

<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("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul>

</div>

</div>

</div>

Ainsi, vous pouvez créer une nouvelle vue partielle. Voyons voir comment rendre vue partielle.

## **Rendu de la vue partielle:**

Vous pouvez rendre la vue partielle de la vue parent à l'aide des méthodes d'aide html: Partiel () ou RenderPartial () ou RenderAction (). Chaque méthode sert à des fins différentes. Ayons un aperçu de chaque méthode, puis voir comment rendre vue partielle en utilisant ces méthodes.

### **Html.Partial ():**

@ Html.Partial () méthode d'aide rend la vue partielle spécifiée. Il accepte partielle nom de vue en tant que paramètre de chaîne et renvoie MvcHtmlString. Il renvoie la chaîne html si vous avez une chance de modifing le html avant de rendre.

Le tableau suivant répertorie les surcharges de la méthode d'assistance partielle:

| **Méthode Helper** | **La description** |
| --- | --- |
| *MvcHtmlString* Html.Partial (string partialViewName) | Rend la vue contenu partiel donné dans la vue visée. |
| *MvcHtmlString* Html.Partial (string partialViewName, modèle d'objet) | Rend la vue partielle du contenu de l'avis visé. paramètre Modèle passe l'objet de modèle à la vue partielle. |
| *MvcHtmlString* Html.Partial (string partialViewName, ViewDataDictionary ViewData) | Rend la vue partielle du contenu de l'avis visé. paramètre View de données passe Voir les données de dictionnaire pour la vue partielle. |
| *MvcHtmlString* Html.Partial (string partialViewName, modèle objet, ViewDataDictionary ViewData) | Rend la vue partielle du contenu de l'avis visé. paramètre Modèle passe l'objet de modèle et afficher les données passe vue du dictionnaire de données à la vue partielle. |

### **Html.RenderPartial ():**

La méthode d'assistance RenderPartial est identique à la méthode partielle, sauf qu'il retourne void et écritures a entraîné html d'une vue partielle spécifiée dans un flux de réponse HTTP directement.

| **méthode d'assistance** | **La description** |
| --- | --- |
| RenderPartial (String partialViewName) | Rend la vue partielle spécifiée |
| RenderPartial (String partialViewName, modèle d'objet) | Rend la vue partielle spécifiée et définir l'objet de modèle spécifié |
| RenderPartial (String partialViewName, ViewDataDictionary ViewData) | Rend la vue partielle spécifiée, remplaçant sa propriété ViewData avec l'objet ViewDataDictionary spécifié. |
| RenderPartial (String partialViewName, modèle objet, ViewDataDictionary ViewData) | Rend la vue partielle spécifiée, remplacer la propriété ViewData de la vue partielle avec l'objet ViewDataDictionary spécifié et définir l'objet de modèle spécifié |

### **Html.RenderAction ():**

La méthode RenderAction helper invoque un contrôleur et l'action spécifiée et rend le résultat comme une vue partielle. La méthode d'action spécifié doit retourner PartialViewResult en utilisant la méthode partielle ().

| **prénom** | **La description** |
| --- | --- |
| RenderAction (String actionName) | Invoque la méthode d'action enfant spécifié et rend le résultat dans la vue parent. |
| RenderAction (String actionName, objet routeValue) | Invoque la méthode d'action enfant spécifié en utilisant les paramètres spécifiés et rend la ligne de résultat dans la vue parent. |
| RenderAction (String actionName, String ControllerName) | Invoque la méthode d'action enfant spécifié en utilisant le nom du contrôleur spécifié et rend la ligne de résultat dans la vue parent. |
| RenderAction (String actionName, routeValues ​​RouteValueDictionary) | Invoque la méthode d'action enfant spécifié en utilisant les paramètres spécifiés et rend la ligne de résultat dans la vue parent. |
| RenderAction (String actionName, String ControllerName, objet routeValue) | Invoque la méthode d'action enfant spécifié en utilisant les paramètres spécifiés et le nom du contrôleur et rend la ligne de résultat dans la vue parent. |
| RenderAction (String actionName, String ControllerName, routeValues ​​RouteValueDictionary) | Invoque la méthode d'action enfant spécifié en utilisant les paramètres spécifiés et le nom du contrôleur et rend la ligne de résultat dans la vue parent. |

Alors maintenant, nous pouvons utiliser l'une des méthodes déchirantes ci-dessus pour rendre la vue partielle \_HeaderNavBar en \_Layout.cshtml. La vue de la mise en page suivante rend vue partielle en utilisant la méthode RenderPartial ().

Exemple: Html.RenderPartial ()

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title - My ASP.NET Application</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

@{

**Html.RenderPartial("\_HeaderNavBar");**

}

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

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

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

**Note:**RenderPartial retourne void, donc un point - virgule est requis à la fin et il doit être placé entre les accolades.

La vue de la mise en page suivante utilise la méthode partielle pour rendre view\_HeaderNavBar.cshtml partielle.

Exemple: Html.Partial ()

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title - My ASP.NET Application</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

**@Html.Partial("\_HeaderNavBar")**

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

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

@Scripts.Render("~/bundles/bootstrap")

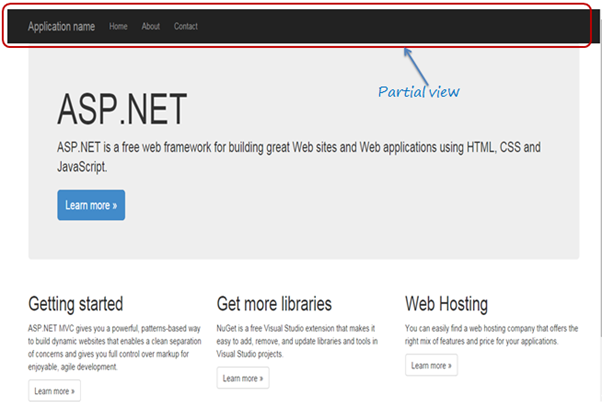
@RenderSection("scripts", required: false)

</body>

</html>

**Remarque:**la méthode @ Html.Partial () n'a pas besoin d'être dans le bloc de code , car il renvoie une chaîne html.

Vous verrez l'interface suivante dans le navigateur lorsque vous exécutez l'application.

[](http://www.tutorialsteacher.com/Content/images/mvc/partial-view-5.png)Index.cshtml

Donc, de cette façon, vous pouvez utiliser une vue partielle sans aucune différence dans l'interface utilisateur.



### **Points à retenir:**

1. Vue partielle est une vue réutilisable, qui peut être utilisé comme une vue de l'enfant dans plusieurs autres points de vue.
2. Vue partielle peut être rendue en utilisant la méthode Html.Partial (), Html.RenderPartial () ou Html.RenderAction ().

### **Lectures complémentaires:**

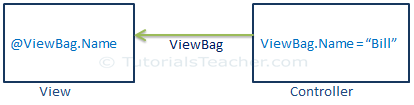
1. [Différence entre html.Partial et Html.RenderPartial.](http://www.tutorialsteacher.com/articles/difference-between-partial-and-renderpartial-mvc)

# ViewBag dans ASP.NET MVC:

Nous avons appris dans la section précédente que l'objet de modèle est utilisé pour envoyer des données en vue de rasoir. Cependant, il peut y avoir un certain scénario où vous voulez envoyer une petite quantité de données temporaires à la vue. Donc, pour cette raison, framework MVC comprend ViewBag.

ViewBag peut être utile lorsque vous souhaitez transférer des données temporaires (qui ne sont pas inclus dans le modèle) à partir du contrôleur à la vue. Le ViewBag est une [dynamique](http://www.tutorialsteacher.com/csharp/csharp-dynamic-type) propriété type de classe ControllerBase qui est la classe de base de tous les contrôleurs.

La figure suivante illustre le ViewBag.

[](http://www.tutorialsteacher.com/Content/images/mvc/viewbag.png)ViewBag

Dans la figure ci-dessus, il attache une propriété Name à ViewBag avec la notation de points et attribue une valeur chaîne "Bill" à elle dans le contrôleur. Cela peut être consulté dans la vue comme @ ViewBag.Name. (@ Est rasoir syntaxe pour accéder à la variable côté serveur.)

http://www.tutorialsteacher.com/Content/images/tips.pngVous pouvez attribuer une primitive ou un objet de type complexe comme une valeur à la propriété ViewBag.

Vous pouvez attribuer un certain nombre de propriétés et valeurs à ViewBag. Si vous attribuez le même nom de propriété à plusieurs reprises pour ViewBag, alors il ne prendra en considération la dernière valeur affectée à la propriété.

**Remarque:**les données ViewBag uniquement les transferts de contrôleur pour voir, pas de visa versa. Les valeurs ViewBag seront nulles si la redirection se produit.

L'exemple suivant montre comment transférer des données à partir du contrôleur pour afficher en utilisant ViewBag.

Exemple: Set ViewBag dans la méthode d'action

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

IList<Student> studentList = new List<Student>() {

new Student(){ StudentID=1, StudentName="Steve", Age = 21 },

new Student(){ StudentID=2, StudentName="Bill", Age = 25 },

new Student(){ StudentID=3, StudentName="Ram", Age = 20 },

new Student(){ StudentID=4, StudentName="Ron", Age = 31 },

new Student(){ StudentID=5, StudentName="Rob", Age = 19 }

};

// GET: Student

public ActionResult Index()

{

ViewBag.TotalStudents = studentList.Count();

return View();

}

}

}

Dans l'exemple ci - dessus, nous voulons afficher le nombre total d'étudiants dans une vue pour la démo. Donc, nous avons joint la propriété TotalStudents au ViewBag et affectées du nombre d'étudiants à l' aide studentList.Count().

Maintenant, dans la vue Index.cshtml, vous pouvez accéder à la propriété ViewBag.TotalStudents et afficher toutes les informations de l'étudiant comme illustré ci-dessous.

Exemple: Acess ViewBag dans une vue

<label>Total Students:</label> @ViewBag.TotalStudents

Sortie:

Total Students: 5

ViewBag ne nécessite pas typecasting tandis retriving valeurs de celle-ci.

En interne, ViewBag est un wrapper autour de [ViewData](http://www.tutorialsteacher.com/mvc/viewdata-in-asp.net-mvc) . Il lancera une exception d'exécution, si le nom de la propriété ViewBag correspond à la clé de ViewData.



### **Points à retenir:**

1. ViewBag transfère les données à partir du contrôleur à la vue, les données idéalement temporaire qui ne figure pas dans un modèle.
2. ViewBag est une propriété dynamique qui tire parti des nouvelles fonctionnalités dynamiques en C # 4.0
3. Vous pouvez affecter un nombre quelconque de propertes et des valeurs à ViewBag
4. La vie de l'ViewBag ne dure que pendant la requête HTTP en cours. Les valeurs ViewBag seront nulles si la redirection se produit.
5. ViewBag est en fait un wrapper autour de ViewData.

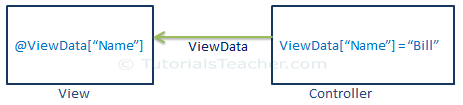
En savoir plus sur ViewData dans la section suivante.

# ViewData:

ViewData est similaire à ViewBag. Il est utile dans le transfert de données à partir du contrôleur à la vue.

ViewData est un dictionnaire qui peut contenir paires clé-valeur où chaque clé doit être la chaîne.

La figure suivante illustre le ViewData.

[](http://www.tutorialsteacher.com/Content/images/mvc/viewdata.png)ViewData

**Remarque:**ViewData seules données transferts à partir du contrôleur pour voir, pas vice-versa. Il est valable uniquement lors de la demande actuelle.

L'exemple suivant montre comment transférer des données à partir du contrôleur pour afficher en utilisant ViewData.

Exemple: ViewData dans la méthode d'action

public ActionResult Index()

{

IList<Student> studentList = new List<Student>();

studentList.Add(new Student(){ StudentName = "Bill" });

studentList.Add(new Student(){ StudentName = "Steve" });

studentList.Add(new Student(){ StudentName = "Ram" });

ViewData["students"] = studentList;

return View();

}

Dans l'exemple ci-dessus, nous avons ajouté une liste des étudiants avec les principaux «étudiants» dans le dictionnaire ViewData. Alors maintenant, la liste des élèves peut être consulté dans une vue comme illustré ci-dessous.

Exemple: Accès ViewData dans un Razor Voir

<ul>

@foreach (var std in ViewData["students"] as IList<Student>)

{

<li>

@std.StudentName

</li>

}

</ul>

S'il vous plaît noter que nous devons jeter des valeurs ViewData le type de données approprié.

Vous pouvez également ajouter un KeyValuePair dans ViewData comme illustré ci-dessous.

Exemple: Ajouter KeyValuePair dans ViewData

public ActionResult Index()

{

ViewData.Add("Id", 1);

ViewData.Add(new KeyValuePair<string, object>("Name", "Bill"));

ViewData.Add(new KeyValuePair<string, object>("Age", 20));

return View();

}

ViewData et ViewBag utilisent tous deux le même dictionnaire interne. Donc, vous ne pouvez pas avoir correspond à ViewData Key avec le nom de la propriété de ViewBag, sinon il va lancer une exception d'exécution.

Exemple: ViewBag et ViewData

public ActionResult Index()

{

ViewBag.Id = 1;

ViewData.Add("Id", 1); // throw runtime exception as it already has "Id" key

ViewData.Add(new KeyValuePair<string, object>("Name", "Bill"));

ViewData.Add(new KeyValuePair<string, object>("Age", 20));

return View();

}



### **Points à retenir:**

1. ViewData transfère les données à partir du contrôleur pour visualiser, pas vice-versa.
2. Viewdata est dérivé d'ViewDataDictionary qui est un type de dictionnaire.
3. La vie de ViewData ne dure que pendant la requête HTTP en cours. valeurs ViewData seront effacées si la redirection se produit.
4. valeur ViewData doit être de type moulé avant utilisation.
5. ViewBag insère à l' intérieur des données dans le dictionnaire ViewData.Donc , la clé de ViewData et la propriété de ViewBag doivent **pas**correspondre.

# TempData:

TempData dans ASP.NET MVC peut être utilisé pour stocker des données temporaires, qui peuvent être utilisés dans la requête suivante. TempData sera effacé après l'achèvement d'une demande ultérieure.

TempData est utile lorsque vous souhaitez transférer des données non sensibles d'une méthode d'action à une autre méthode d'action du même ou d' un autre contrôleur, ainsi que des redirections.Il est de type dictionnaire qui est dérivé de [TempDataDictionary](https://msdn.microsoft.com/en-us/library/system.web.mvc.tempdatadictionary(v=vs.118).aspx) .

Vous pouvez ajouter une paire clé-valeur dans TempData comme le montre l'exemple ci-dessous.

Exemple: TempData

public class HomeController : Controller

{

// GET: Student

public HomeController()

{

}

public ActionResult Index()

{

TempData["name"] = "Test data";

TempData["age"] = 30;

return View();

}

public ActionResult About()

{

string userName;

int userAge;

if(TempData.ContainsKey("name"))

userName = TempData["name"].ToString();

if(TempData.ContainsKey("age"))

userAge = int.Parse(TempData["age"].ToString());

// do something with userName or userAge here

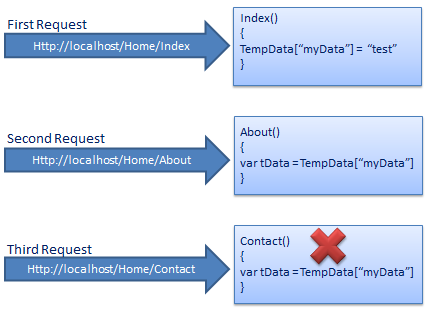
return View();

}

}

Dans l'exemple ci-dessus, nous avons ajouté des données dans TempData et accéder aux mêmes données à l'aide d'une clé dans un autre mode d'action. S'il vous plaît noter que nous avons converti les valeurs dans le type approprié.

La figure suivante illustre TempData.

[](http://www.tutorialsteacher.com/Content/images/mvc/tempdata.png)TempData

http://www.tutorialsteacher.com/Content/images/tips.pngTempData utilise en interne session pour stocker les données.Donc, les données doivent être sérialisés si vous décidez de passer loin de la Session-State mode par défaut et utilisez le mode serveur État ou le mode SQL Server.

Comme vous pouvez le voir dans l'exemple ci-dessus, nous ajoutons des données de test en TempData dans la première demande et dans la deuxième demande ultérieure nous accédons données de test à partir de TempData que nous stockons dans la première demande. Cependant, vous ne pouvez pas obtenir les mêmes données dans la troisième requête parce TempData sera effacé après la deuxième demande.

Appelez TempData.Keep () pour conserver les valeurs TempData dans une troisième demande consécutive.

Exemple: TempData.Keep ()

public class HomeController : Controller

{

// GET: Student

public HomeController()

{

}

public ActionResult Index()

{

TempData["myData"] = "Test data";

return View();

}

public ActionResult About()

{

string data;

if(TempData["myData"] != null)

data = TempData["myData"] as string;

TempData.Keep();

return View();

}

public ActionResult Contact()

{

string data;

if(TempData["myData"] != null)

data = TempData["myData"] as string;

return View();

}

}



### **Points à retenir:**

1. TempData peut être utilisé pour stocker des données entre deux requêtes successives. Les valeurs TempData seront conservées lors de la redirection.
2. TemData est un type de TempDataDictionary.
3. TempData internaly utiliser session pour stocker les données. Alors, pensez-y comme une courte session vécu.
4. valeur TempData doit être de type moulé avant utilisation. Vérifiez les valeurs NULL pour éviter les erreurs d'exécution.
5. TempData peut être utilisé pour stocker seulement un message de temps comme des messages d'erreur, les messages de validation.
6. Appel TempData.Keep () pour conserver toutes les valeurs de TempData dans une troisième demande.

# Filtres à MVC:

Dans ASP.NET MVC, une demande de l'utilisateur est dirigé vers la méthode du contrôleur et des mesures appropriées. Cependant, il peut y avoir des circonstances où vous voulez exécuter une certaine logique avant ou après une méthode d'action exécute. ASP.NET MVC fournit des filtres à cet effet.

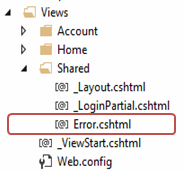
ASP.NET MVC Filter est une classe personnalisée où vous pouvez écrire une logique personnalisée à exécuter avant ou après une méthode d'action exécute. Les filtres peuvent être appliqués à une méthode d'action ou d'un contrôleur de manière déclarative ou programmatique. des moyens déclaratives en appliquant un attribut de filtre à une méthode d'action ou de classe de contrôleur et des moyens de programmation en mettant en œuvre une interface correspondante.

MVC fournit différents types de filtres. La liste de table types de filtres suivants, filtres intégrés pour le type et l'interface qui doit être mis en œuvre pour créer une classe de filtre personnalisé.

| **Type de filtre** | **La description** | **Built-in Filter** | **Interface** |
| --- | --- | --- | --- |
| les filtres d'autorisation | Effectue l'authentification et autorise avant d'exécuter la méthode d'action. | [Authorize], [RequireHttps] | IAuthorizationFilter |
| filtres d'action | Effectue une opération avant et après une méthode d'action exécute. |  | IActionFilter |
| filtres de résultat | Effectue une opération avant ou après l'exécution de vue résultat. | [OutputCache] | IResultFilter |
| les filtres d'exception | Effectue une opération s'il y a une exception non gérée levée lors de l'exécution du pipeline ASP.NET MVC. | [HandleError] | IExceptionFilter |

Pour comprendre le filtre en détail, nous allons prendre un exemple de filtre d'exception intégré.

Un filtre d'exception exécute quand il y a une exception non gérée se produit dans votre application.HandleErrorAttribute ([HandlerError]) classe est une exception classe de filtre intégré dans framework MVC. Cette built-in HandleErrorAttribute classe rend Error.cshtml inclus dans le dossier partagé par défaut, quand une exception non gérée se produit.

[](http://www.tutorialsteacher.com/Content/images/mvc/errorpage.png)Error.cshtml

L'exemple suivant illustre intégré dans le filtre d'exception HandErrorAttribute.

Exemple de filtre d'autorisation:

[HandleError]

public class HomeController : Controller

{

public ActionResult Index()

{

//throw exception for demo

throw new Exception("This is unhandled exception");

return View();

}

public ActionResult About()

{

return View();

}

public ActionResult Contact()

{

return View();

}

}

http://www.tutorialsteacher.com/Content/images/tips.pngChaque classe d'attribut doit se terminer avec l'attribut par exemple HanderErrorAttribute.L'attribut doit être appliqué sans suffixe d'attribut entre crochets [] comme [HandelError].

Dans l'exemple ci - dessus, nous avons appliqué[HandleError]attribut à HomeController. Alors maintenant , il affichera la page d'erreur si une méthode de HomeController d'action jetterait exception non gérée. S'il vous plaît noter que exception non gérée est une exception qui ne sont pas pris encharge par le bloc try-catch.

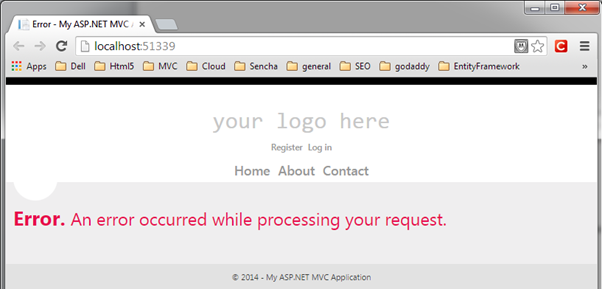
Les filtres appliqués au contrôleur seront automatiquement applicables à toutes les méthodes d'un contrôleur d'action.

faire S'il vous plaît vous que le mode CustomError est dans la section System.web de web.config, pour HandleErrorAttribute fonctionne correctement.

CustomError dans web.config:

<customErrors mode="On" />

Maintenant, si vous exécutez l'application. Vous obtiendrez une page d'erreur suivante parce que nous jetons exception dans la méthode d'action Index aux fins de démonstration.

[](http://www.tutorialsteacher.com/Content/images/mvc/handleerror-demo.png)HandleError Demo

Ainsi, l'attribut HandleError affichera la page d'erreur commune pour toute exception non gérée est survenue dans HomeController.

## **Inscription Filtres:**

Les filtres peuvent être appliqués à trois niveaux.

### **1. Niveau mondial:**

Vous pouvez appliquer des filtres au niveau mondial en cas de Application\_Start fichier Global.asax.cs en utilisant FilterConfig.RegisterGlobalFilters par défaut () mehtod. Les filtres globaux seront appliqués à toutes les méthodes d'une application contrôleur et d'action.

Le [HandleError] filtre est appliqué globaly dans MVC application par défaut dans chaque application MVC créé en utilisant Visual Studio comme illustré ci-dessous.

Exemple: Enregistrer Filtres de niveau mondial

// MvcApplication class contains in Global.asax.cs file

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

}

}

// FilterConfig.cs located in App\_Start folder

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

filters.Add(new HandleErrorAttribute());

}

}

### **2. Niveau Controller:**

Les filtres peuvent également être appliqués à la classe du contrôleur. Ainsi, les filtres seront applicables à tous la méthode de la classe contrôleur d'action si elle est appliquée à une classe de contrôleur.

Exemple: Filtres d'action au contrôleur

[HandleError]

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

}

### **3. Méthode d'action niveau:**

Vous pouvez appliquer des filtres à une méthode d'action individuelle aussi. Ainsi, le filtre sera applicable à cette méthode d'action particulière seulement.

Appliquer le filtre à la méthode d'action:

public class HomeController : Controller

{

[HandleError]

public ActionResult Index()

{

return View();

}

}

De la même façon, vous pouvez appliquer plusieurs filtres à construire ou personnalisés au niveau mondial ou au niveau du contrôleur ou méthode d'action à des fins différentes, comme [Authorize], [RequireHttps], [ChildActionOnly], [OutputCache], [HandleError].

## **Filtre Trier:**

Comme mentionné ci-dessus, MVC comprend différents types de filtres et de multiples filtres peuvent être appliqués à une méthode de classe ou de l'action seul contrôleur. Ainsi, les filtres exécutés dans l'ordre suivant.

1. les filtres d'autorisation
2. filtres d'action
3. les filtres de réponse
4. les filtres d'exception

## **Créer un filtre personnalisé:**

Vous pouvez créer des attributs de filtre personnalisé en mettant en œuvre une interface de filtre approprié pour lequel vous voulez créer un filtre personnalisé et tirez également une classe FilterAttribute de sorte que vous pouvez utiliser cette classe comme un attribut.

Par exemple, la mise en œuvre IExceptionFilter et FilterAttribute classe pour créer filtre d'exception personnalisée. De la même façon de mettre en œuvre une interface de classe et IAuthorizatinFilter FilterAttribute pour créer un filtre d'autorisation sur mesure.

Exemple: Filtre d'exception personnalisée

class MyErrorHandler : FilterAttribute, IExceptionFilter

{

public override void IExceptionFilter.OnException(ExceptionContext filterContext)

{

Log(filterContext.Exception);

base.OnException(filterContext);

}

private void Log(Exception exception)

{

//log exception here..

}

}

Alternativement, vous pouvez également dériver une classe de filtre intégré et remplacer une méthode appropriée pour étendre les fonctionnalités de filtres intégrés.

Créons filtre d'exception personnalisée pour enregistrer tous exception non gérée par dérivation intégré dans la classe HandleErrorAttribute et dominante OnException méthode comme indiqué ci-dessous.

Exemple: Filtre d'exception personnalisée

class MyErrorHandler : HandleErrorAttribute

{

public override void OnException(ExceptionContext filterContext)

{

Log(filterContext.Exception);

base.OnException(filterContext);

}

private void Log(Exception exception)

{

//log exception here..

}

}

Maintenant, vous pouvez appliquer l'attribut MyErrorHandler au niveau mondial ou d'un contrôleur ou d'une méthode d'action niveau, de la même façon, nous avons appliqué l'attribut HandleError.

Exemple: mesure d'action Filtres à Controller

[MyErrorHandler]

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

}



### **Points à retenir:**

1. Filtres MVC sont utilisés pour exécuter une logique personnalisée avant ou après l'exécution de la méthode d'action.
2. MVC Filter types:
   1. les filtres d'autorisation
   2. filtres d'action
   3. filtres de résultat
   4. les filtres d'exception
3. Les filtres peuvent être appliqués globalement en classe FilterConfig, au niveau du contrôleur ou de l'action du niveau de la méthode.
4. classe de filtre personnalisée peut être créée par la mise en œuvre de classe FilterAttribute et interface correspondante.

En savoir plus sur Filtres d'action dans la section suivante.

# Filtres d'action:

Dans la section précédente, vous avez appris sur les filtres dans MVC. Dans cette section, vous apprendrez un autre type de filtre appelé **Filtres d'action** dans ASP.NET MVC.

filtre d'action exécute avant et après une méthode d'action exécute. les attributs de filtre d'action peuvent être appliquées à un mode d'action particulier ou à un contrôleur. Lorsque le filtre d'action appliquée au contrôleur, puis il sera appliqué à toutes les méthodes d'action dans ce contrôleur.

OutputCache est un attribut de filtre d'action intégré qui peut être applicable à une méthode d'action pour laquelle nous voulons mettre en cache la sortie. Par exemple, la sortie de la méthode d'action suivante sera mise en cache pendant 100 secondes.

Exemple: ActionFilter

[OutputCache(Duration=100)]

public ActionResult Index()

{

return View();

}

Vous pouvez créer le filtre d'action personnalisé pour votre application. Voyons voir comment créer des filtres d'action personnalisé.

## **Action personnalisée Filtre:**

Vous pouvez créer le filtre d'action personnalisé de deux manières. Tout d'abord, en mettant en œuvre l'interface IActionFilter et la classe FilterAttribute. Deuxièmement, en dérivant ActionFilterAttribute classe abstraite.

Interface IActionFilter comprennent des méthodes suivantes pour mettre en œuvre:

* vide OnActionExecuted (ActionExecutedContext filterContext)
* OnActionExecuting void (ActionExecutingContext filterContext)

*ActionFilterAttribute* classe abstraite comprend les méthodes suivantes pour passer outre:

* vide OnActionExecuted (ActionExecutedContext filterContext)
* OnActionExecuting void (ActionExecutingContext filterContext)
* vide OnResultExecuted (ResultExecutedContext filterContext)
* OnResultExecuting void (ResultExecutingContext filterContext)

Comme vous pouvez voir que la classe ActionFilterAttribute a quatre méthodes de surcharger. Il comprend OnResultExecuted et OnResultExecuting méthodes, qui peuvent être utilisés pour exécuter une logique personnalisée avant ou après résultat exécute. les filtres d'action sont généralement utilisés pour appliquer des préoccupations transversales telles que l'exploitation forestière, la mise en cache, l'autorisation etc.

Considérez la coutume Log classe de filtre suivant pour l'exploitation forestière.

Exemple: ActionFilter personnalisée pour l'exploitation forestière

public class LogAttribute : ActionFilterAttribute

{

public override void OnActionExecuted(ActionExecutedContext filterContext)

{

Log("OnActionExecuted", filterContext.RouteData);

}

public override void OnActionExecuting(ActionExecutingContext filterContext)

{

Log("OnActionExecuting", filterContext.RouteData);

}

public override void OnResultExecuted(ResultExecutedContext filterContext)

{

Log("OnResultExecuted", filterContext.RouteData);

}

public override void OnResultExecuting(ResultExecutingContext filterContext)

{

Log("OnResultExecuting ", filterContext.RouteData);

}

private void Log(string methodName, RouteData routeData)

{

var controllerName = routeData.Values["controller"];

var actionName = routeData.Values["action"];

var message = String.Format("{0}- controller:{1} action:{2}", methodName,

controllerName,

actionName);

Debug.WriteLine(message);

}

}

Comme vous pouvez le voir, la classe Log classe dérivée ActionFilterAttribute. Il enregistre avant et après la méthode d'action ou le résultat exécute. Vous pouvez appliquer l'attribut journal pour toutes les méthodes de contrôleur ou d'action où vous souhaitez enregistrer l'action. Par exemple, en appliquant l'attribut Log Controller, il enregistre chaque action les méthodes de ce contrôleur.

Exemple: Appliquer ActionFilter Connexion au contrôleur

[Log]

public class StudentController : Controller

{

public ActionResult Index()

{

return View();

}

public ActionResult About()

{

return View();

}

public ActionResult Contact()

{

return View();

}

}

L'exemple ci - dessus affichera la sortie suivante dans la fenêtre de sortie de Visual Studio sur *http: // localhost / étudiant* demande.

Sortie:

OnActionExecuting- controller:Home action:Index

OnActionExecuted- controller:Home action:Index

OnResultExecuting - controller:Home action:Index

OnResultExecuted- controller:Home action:Index



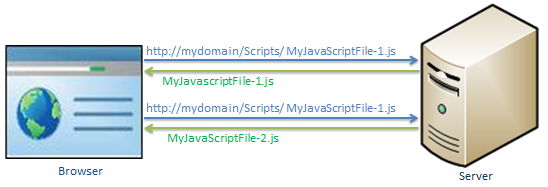
### **Points à retenir:**

1. les filtres d'action permettent de pré et de la logique post-traitement à appliquer à une méthode d'action.
2. les filtres d'action sont généralement utilisés pour appliquer des préoccupations transversales telles que l'exploitation forestière, la mise en cache, l'autorisation etc.
3. filtre d'action peut être enregistré comme autres filtres au niveau mondial, le contrôleur ou l'action du niveau de la méthode.
4. attribut de filtre d'action personnalisée peut être créée en dérivant la classe ActionFilterAttribute ou mise en œuvre de l'interface IActionFilter et FilterAttribute classe abstraite.
5. Chaque filtre d'action doit remplacer OnActionExecuted, OnActionExecuting, OnResultExecuted, OnResultExecuting méthodes.

# Bundling:

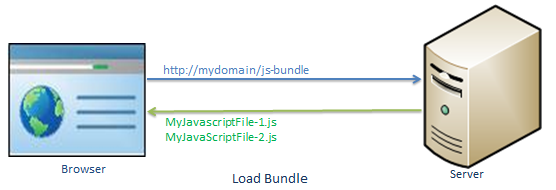
techniques de Bundling et minification ont été introduits dans MVC 4 pour améliorer la demande de temps de charge. Paquetteurs nous permettent de charger le groupe de fichiers statiques à partir du serveur en une seule requête HTTP.

La figure suivante illustre les techniques de groupement:

[](http://www.tutorialsteacher.com/Content/images/mvc/bundling-1.png)fichiers de script de charge dans des requêtes distinctes

Dans la figure ci-dessus, le navigateur envoie deux demandes distinctes pour charger deux fichiers JavaScript MyJavaScriptFile-1.js différents et MyJavaScriptFile-2.js.

Paquetteurs technique dans MVC 4 nous permet de charger plus d'un fichier JavaScript, MyJavaScriptFile-1.js et MyJavaScriptFile-2.js dans une requête comme illustré ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/bundling-2.png)

## **minification:**

technique Minification optimise la taille du fichier de script ou css en supprimant l'espace et les commentaires blancs inutiles et de raccourcir les noms de variables à un seul caractère.

Par exemple, considérons la fonction suivante JavaScript.

Exemple: JavaScript

sayHello = function(name){

//this is comment

var msg = "Hello" + name;

alert(msg);

}

Le ci-dessus JavaScript seront optimisés et réduits au minimum dans le script suivant.

Exemple: minified JavaScript

sayHello=function(n){var t="Hello"+n;alert(t)}

Comme vous pouvez le voir ci-dessus, il a retiré inutiles espace blanc, des commentaires et de raccourcir également les noms de variables pour réduire les caractères qui à son tour permettra de réduire la taille du fichier JavaScript.

impacts Bundling et minification sur le chargement de la page, il charge la page plus rapidement en minimisant la taille du fichier et le nombre de demandes.

## **Types Bundle:**

MVC 5 comprend les classes suivantes bundle dans System.web.Optimization espace de noms:

**ScriptBundle** : ScriptBundle est responsable de JavaScript minification des fichiers de script simples ou multiples.

**StyleBundle** : StyleBundle est responsable de la CSS minification des fichiers de feuille de style unique ou multiple.

**DynamicFolderBundle** : Représente un objet Bundle que ASP.NET crée à partir d' un dossier qui contient les fichiers du même type.

Toutes les classes de faisceaux ci - dessus sont inclus dans *System.Web.Optimization.Bundle* espace de noms et dérivés de la [classe Bundle](http://msdn.microsoft.com/en-us/library/system.web.optimization.bundle(v=vs.110).aspx) .

En savoir plus sur ScriptBundle dans la section suivante.



### **Points à retenir:**

1. Bundling et Minification minimisent les fichiers de script ou css statiques temps therby minimiser la page du temps de chargement de chargement.
2. framework MVC fournit des classes ScriptBundle, StyleBundle et DynamicFolderBundle.
3. ScriptBundle fait minification des fichiers JavaScript.
4. StyleBundle fait minification des fichiers CSS.

# ScriptBundle dans ASP.NET MVC:

Nous avons appris comment le regroupement technique fonctionne dans ASP.NET MVC. Ici, nous allons apprendre à créer un faisceau de plusieurs fichiers JavaScript dans une requête HTTP.

ASP.NET MVC API inclut [ScriptBundle](https://msdn.microsoft.com/en-us/library/system.web.optimization.scriptbundle(v=vs.110).aspx) classe qui minification JavaScript et le regroupement.

Ouvrez le fichier App\_Start \ BundleConfig.cs dans les dossiers MVC. Le fichier BundleConfig.cs est créé par MVC cadre par défaut. Vous devriez écrire tout le code de votre regroupement dans la méthode BundleConfig.RegisterBundles (). (Vous pouvez créer votre propre classe personnalisée au lieu d'utiliser la classe BundleConfig, mais il est recommandé de suivre la pratique standard.) Le code suivant montre une partie de la méthode RegisterBundles.

Exemple: BundleConfig.RegisterBundle ()

using System.Web;

using System.Web.Optimization;

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

// create an object of ScriptBundle and

// specify bundle name (as virtual path) as constructor parameter

ScriptBundle scriptBndl = new ScriptBundle("~/bundles/bootstrap");

//use Include() method to add all the script files with their paths

scriptBndl.Include(

"~/Scripts/bootstrap.js",

"~/Scripts/respond.js"

);

//Add the bundle into BundleCollection

bundles.Add(scriptBndl);

BundleTable.EnableOptimizations = true;

}

}

Dans l'exemple ci-dessus, nous avons créé un paquet de deux fichiers JavaScript, bootstrap.js et respond.js utilisant ScriptBundle à des fins de démonstration.

1. Tout d'abord créer une instance de la classe ScriptBundle en spécifiant le nom de l'ensemble comme un paramètre du constructeur. Ce nom de paquet est un chemin d'accès virtuel à partir de ~ /. Vous pouvez donner quelque chose en chemin virtuel, mais il est recommandé de donner un chemin qui sera facile à identifier comme un paquet. Ici, nous avons donné "/ bundles / ~ bootstrap" chemin, afin que nous puissions identifier facilement que ce paquet inclut les fichiers associés bootstrap.
2. Utilisez Inclure méthode pour ajouter un ou plusieurs fichiers JS dans un bundle avec le chemin relatif après chemin racine en utilisant ~ signe.
3. Finale, ajouter le paquet dans exemple BundleCollection, qui est spécifié comme paramètre dans RegisterBundle (méthode).
4. Dernière, BundleTable.EnableOptimizations = truepermet de regrouper et de minification en mode debug. Si vous le définissez sur false, il ne fera pas le regroupement et minification.

Vous pouvez également utiliser IncludeDirectory la méthode de classe de faisceau pour ajouter tous les fichiers sous le répertoire particulier, comme indiqué ci-dessous.

Exemple ScriptBundle:

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/scripts").IncludeDirectory("~/Scripts/","\*.js",true));

}

Ainsi, vous pouvez créer un ensemble de fichiers JavaScript en utilisant ScriptBundle. framework MVC appelle la méthode BundleConfig.RegisterBundle () de l'événement Application\_Start dans le fichier Global.asax.cs, de sorte qu'il peut ajouter tous les faisceaux en BundleCollection au départ d'une application.

Exemple: Invoke RegisterBundle () dans l'événement Application\_Start

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

## **Utilisation de caractères génériques:**

Parfois, les fichiers de script tiers comprend des versions en un nom de fichier de script. Donc, il ne convient pas à des changements du code à chaque fois que vous mettez à niveau la version du fichier de script. Avec l'utilisation de caractères génériques, vous ne devez pas spécifier une version d'un fichier de script. Il inclut automatiquement les fichiers avec la version disponible.

Par exemple, les fichiers Jquery inclut la version dans un nom. Ainsi, vous pouvez utiliser {version} générique pour pickup une version basée sur la version disponible.

Exemple: Wildcard avec bundle

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery")

.Include( "~/Scripts/jquery-{version}.js"));

}

}

Maintenant, il va ramasser fichier ajouté dans un projet jQuery. Si vous avez inclus jquery-1.7.1.js alors il rendra ce fichier et lorsque vous mettez à jour le fichier de jQuery pour jquery-1.10.2.js alors il sera automatiquement rendre le fichier 1.10 version sans modifier ni la compilation du code.

## **Utilisation de CDN:**

Vous pouvez également utiliser Content Delivery Network pour charger les fichiers de script. Par exemple, vous pouvez charger la bibliothèque jquery de CDN comme indiqué ci-dessous.

Exemple: Charger des fichiers de CDN

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

var cdnPath = "http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.1.min.js";

bundles.Add(new ScriptBundle("~/bundles/jquery", cdnPath)

.Include( "~/Scripts/jquery-{version}.js"));

}

}

Dans le code ci-dessus, jquery sera demandé au tout CDN en mode de libération et dans le mode de débogage, la bibliothèque jQuery sera chargée à partir d'une source locale. S'il vous plaît noter que vous devez disposer d'un mécanisme de repli pour faire face à un échec de demande CDN.

Maintenant, nous allons voir comment utiliser le faisceau dans une vue de rasoir.

## **Inclure ScriptBundle dans Razor Vue:**

Nous avons créer un faisceau de script ci-dessus. Maintenant, nous allons apprendre comment inclure bundle en vue de rasoir.

Les faisceaux de script peuvent être inclus en utilisant statique [Scripts](http://msdn.microsoft.com/en-us/library/system.web.optimization.scripts(v=vs.110).aspx) classe. Utiliser la méthode Scripts.Render () pour inclure spécifié paquet de script lors de l' exécution.

Exemple: Scripts.Render ()

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title</title>

**@Scripts.Render("~/bundles/bootstrap")**

</head>

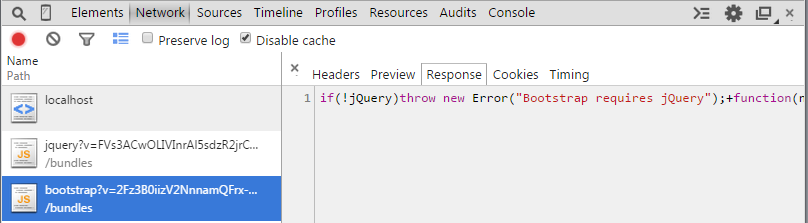
<body>

@\*html code removed for clarity \*@

</body>

</html>

Maintenant, si vous exécutez l'exemple ci-dessus, vous trouverez deux fichiers de script est combiné, minified et chargées en une seule demande. faire S'il vous plaît assurer que vous avez défini debug = false dans web.config<compilation debug="false" targetFramework="4.5"/>

[](http://www.tutorialsteacher.com/Content/images/mvc/bundling-3.png)Charger Bundle dans le navigateur

Comme vous pouvez le voir dans la figure ci-dessus ce paquet bootstrap est chargé en une seule demande. Il a également combiné et minified deux fichiers JS pour bootstrap.



### **Points à retenir:**

1. Bundling et Minification minimisent les fichiers de script ou css statiques temps therby minimiser la page du temps de chargement de chargement.
2. ScriptBundle fait minification des fichiers JavaScript.
3. Créer un script ou css faisceaux en classe BundleConfig inclus dans le dossier App\_Start.
4. Utilisez joker {version} pour rendre la version disponible des fichiers lors de l'exécution.
5. Utiliser la méthode Scripts.Render ( "nom bundle") pour inclure faisceau de script dans une vue de rasoir.

# StyleBundle:

Vous avez appris comment créer un ensemble de fichiers JavaScript dans la section précédente. Ici, vous apprendrez à créer un ensemble de fichiers de feuille de style (CSS).

API ASP.NET MVC comprend [StyleBundle](https://msdn.microsoft.com/en-us/library/system.web.optimization.stylebundle(v=vs.110).aspx) classe qui CSS minification et le regroupement. StyleBundle est également dérivée d'une classe Bundle pour qu'il supporte mêmes méthodes que ScriptBundle.

Comme mentionné dans la section précédente, vous devez créer des paquets de fichiers de script et css dans les RegisterBundles () la méthode de classe BundleConfig contenue dans App\_Start -> fichier BundleConfig.cs.

Le code suivant montre une partie des RegisterBundles (méthode).

Utilisez ScriptsInclude ou méthode IncludeDerictory pour ajouter des fichiers css dans bundle comme indiqué ci-dessous:

Exemple: StyleBundle

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new StyleBundle("~/bundles/css").Include(

"~/Content/bootstrap.css",

"~/Content/site.css"

));

// add ScriptBundle here..

}

}

Comme vous pouvez le voir dans l'exemple ci-dessus, nous avons créé exemple StyleBundle avec le nom liasse sous chemin virtuel. Le nom de l'ensemble (chemin virtuel) doit commencer par ~ /.Utilisez Include () ou la méthode IncludeDirectory () avec des noms de fichier css sous forme de chaîne.

Vous pouvez utiliser générique et le chemin CDN de la même manière que ScriptBundle comme indiqué dans la section précédente.

## **Inclure style Bundle dans Razor Voir:**

Vous pouvez utiliser StyleBundle dans une vue de mise en page et de rendre tas de fichiers css dans une requête unique à l' aide statique [Styles](http://msdn.microsoft.com/en-us/library/system.web.optimization.styles(v=vs.110).aspx) classe. Styles est une classe d'aide pour rendre lesfaisceaux de css.

Exemple: Inclure style Bundle Voir

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title - My ASP.NET Application</title>

**@Styles.Render("~/bundles/css")**

</head>

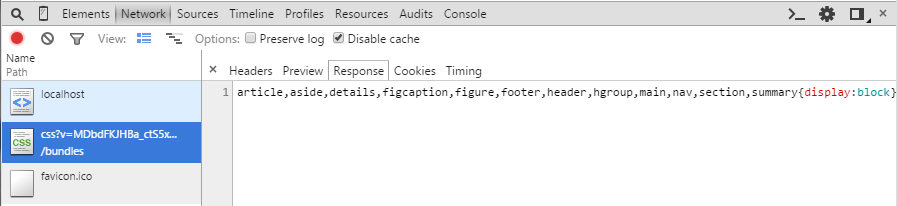
<body>

@\*html code removed for clarity \*@

</body>

</html>

Comme le montre l'exemple ci-dessus, utilisez la méthode Styles.Render () pour inclure spécifié faisceau de css lors de l'exécution. Ouvrir outil de développement du navigateur et vérifier qu'il a minified et chargé des fichiers css comme indiqué ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/stylebundle-demo.png)Charger Bundle dans le navigateur

### **Lectures complémentaires:**

* [Comment configurer chemin de l'image dans StyleBundle.](http://www.tutorialsteacher.com/articles/how-to-set-image-path-in-style-bundle)



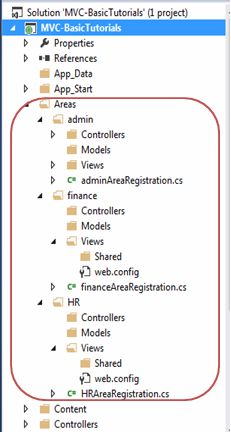
### **Points à retenir:**

1. Bundling et Minification minimisent les fichiers de script ou css statiques temps therby minimiser la page du temps de chargement de chargement.
2. framework MVC fournit des classes ScriptBundle, StyleBundle et DynamicFolderBundle.
3. StyleBundle fait minification des fichiers CSS.
4. Créer un script ou css faisceaux dans la classe BundleConfig inclus dans le dossier App\_Start.
5. Utilisez joker {version} pour rendre la version disponible des fichiers lors de l'exécution.
6. Utiliser la méthode Styles.Render ( "nom bundle") pour inclure des faisceaux de style dans une vue de rasoir.

# Région:

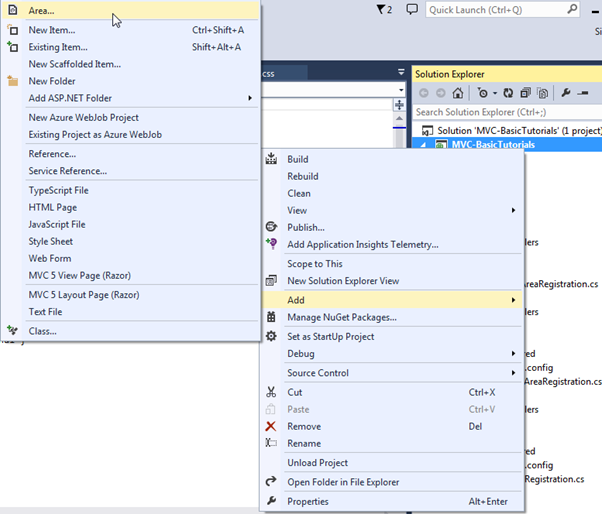
Vous avez déjà appris que le framework ASP.NET MVC comprend des dossiers séparés pour le modèle, vue et le contrôleur. Cependant, une grande application peut inclure un grand nombre de contrôleurs, vues et classes de modèle. Donc, pour maintenir un grand nombre de vues, des modèles et des contrôleurs avec la structure du projet ASP.NET MVC par défaut peut devenir ingérable

ASP.NET MVC 2 introduit la région. Zone nous permet de partitionner grande application dans des unités plus petites où chaque unité contient structure séparée du dossier MVC, même que la structure des dossiers par défaut MVC. Par exemple, une grande application d'entreprise peut avoir différents modules comme admin, finance, ressources humaines, marketing, etc. Donc, une zone peut contenir la structure de dossier MVC séparé pour tous ces modules comme indiqué ci-dessous.

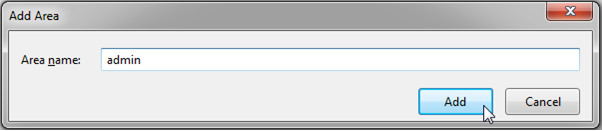
[](http://www.tutorialsteacher.com/Content/images/mvc/area-1.png)Région

## **Créer une zone:**

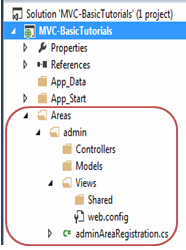
Vous pouvez créer une zone en utilisant ASP.NET MVC 5 et Visual Studio 2013 pour web en cliquant droit sur le projet dans l'explorateur de solution -> Ajouter -> Zone ..

[](http://www.tutorialsteacher.com/Content/images/mvc/create-area.png)Région

Entrez Nom de la zone dans la boîte de dialogue Ajouter une zone et cliquez sur Ajouter.

[](http://www.tutorialsteacher.com/Content/images/mvc/area-2.png)Région

Cela va ajouter le dossier 'admin' sous le dossier Zone comme illustré ci-dessous.

[](http://www.tutorialsteacher.com/Content/images/mvc/area-3.png)Région

Comme vous pouvez le voir, chaque zone comprend la classe de AreaRegistration dans {nom de domaine} + fichier AreaRegistration.cs.

Ce qui suit est classe adminAreaRegistration créé avec zone d'administration.

Inscription:

public class adminAreaRegistration : AreaRegistration

{

public override string AreaName

{

get

{

return "admin";

}

}

public override void RegisterArea(AreaRegistrationContext context)

{

context.MapRoute(

"admin\_default",

"admin/{controller}/{action}/{id}",

new { action = "Index", id = UrlParameter.Optional }

);

}

}

Classe AreaRegistration remplace la méthode RegisterArea pour cartographier les routes de la zone.Dans l'exemple ci - dessus, toute URL qui commence par **administration** sera assurée par les contrôleurs inclus dans la structure du dossier d'administration sous le dossier Zone. Par exemple,*http: // localhost / admin / profil* sera traitée par le contrôleur de profil inclus dans les zones / admin / contrôleur dossier / ProfileController.

Enfin, toute la zone doit être enregistré dans l'événement Application\_Start en Global.asax.cs comme AreaRegistration.RegisterAllAreas();

Donc, de cette façon, vous pouvez créer et maintenir de multiples domaines pour la grande application.