TP ASP MVC

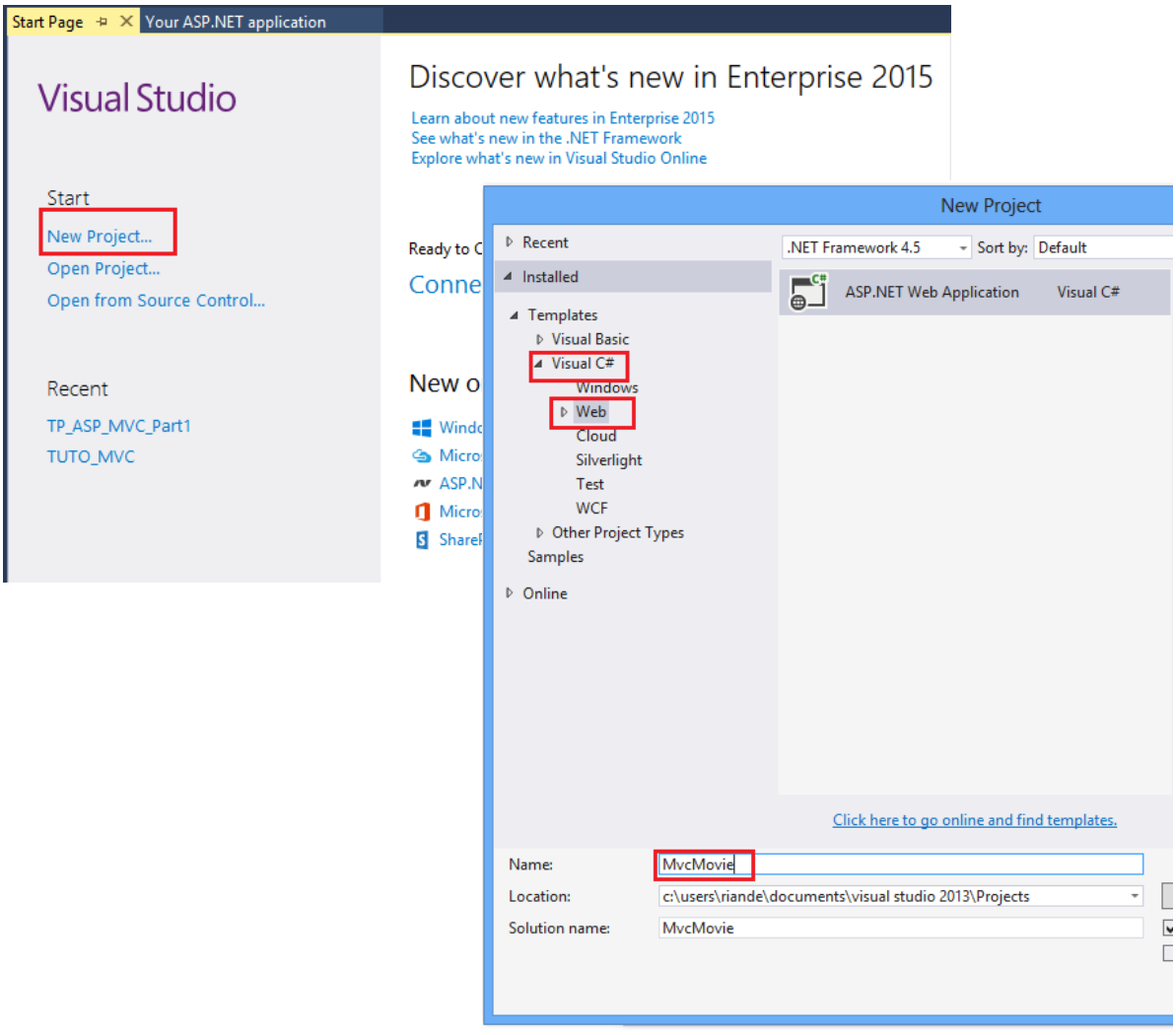
MVC est synonyme de *modèle-vue-contrôleur*. MVC est un pattern pour développer des applications qui sont bien architecturé, testables et facile à entretenir. MVC veut dire :

- **M**odel : Classes qui représentent les données de l'application et qui utilisent une logique de validation pour faire respecter les règles de métier.
- **V**iew : fichiers de modèle que votre application utilise pour générer dynamiquement les réponses HTML.
- **C**ontroller : Classes qui gèrent les demandes entrantes de navigateur, récupérer des données de modèle et ensuite spécifier des modèles de vue qui retournent une réponse au navigateur.

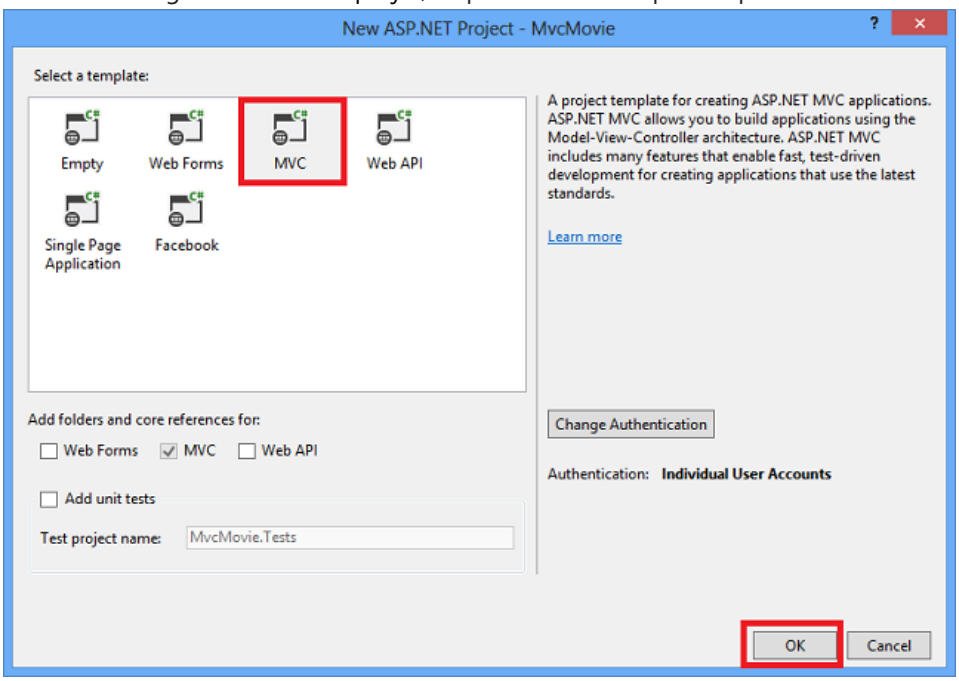
Nous couvrons tous ces concepts dans cette série de tutoriel et vous montrons comment s'en servir pour construire une application Web.

Création d'un projet ASP MVC

1. Cliquez sur **New Project**, puis sélectionnez Visual c# sur la gauche, puis **Web** et puis sélectionnez **ASP.NET Application Web**. Nommez votre projet «**MvcMovie**» et puis cliquez sur **OK**.

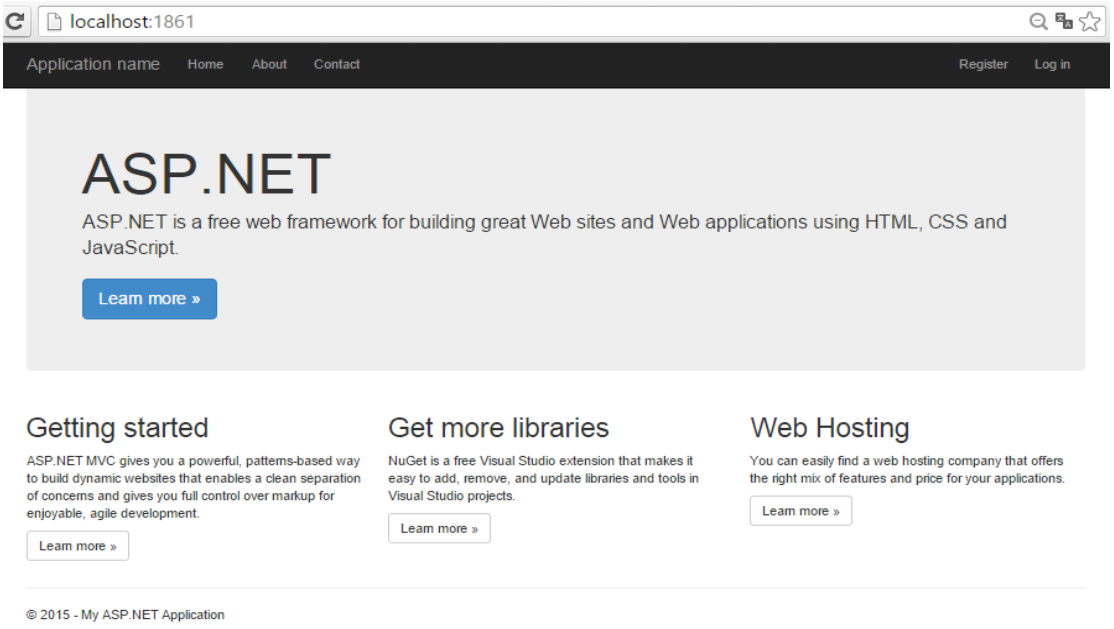


2. Dans la boîte de dialogue du nouveau projet, cliquez sur **MVC** et puis cliquez sur **OK**.



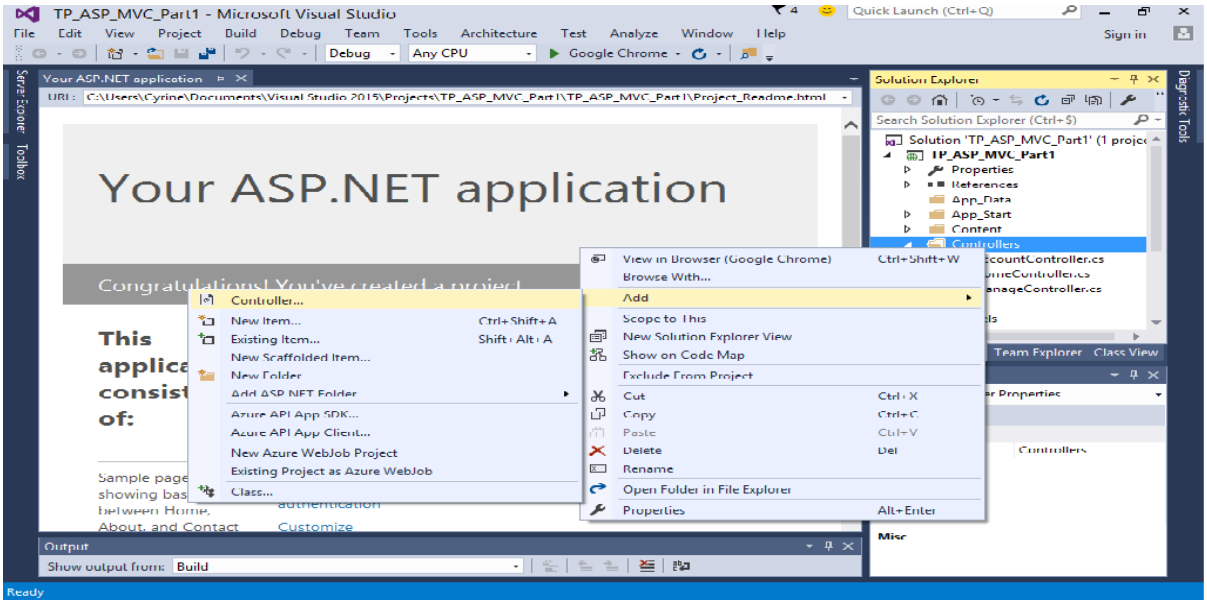
3. Cliquez sur F5 pour démarrer le débogage. F5 oblige Visual Studio à lancer IIS Express et exécuter votre application web Visual Studio puis lance un navigateur et ouvre la page d'accueil de l'application.

Comme vous remarquez la barre d'adresse du navigateur contient : **localhost:port#** et non pas quelque chose comme **example.com**. C'est parce que **localhost** pointe toujours vers votre ordinateur local, qui dans ce cas exécute l'application que vous venez de créer. Lorsque Visual Studio exécute un projet web, un port aléatoire est utilisé pour le serveur web. Dans l'image ci-dessus, le numéro de port est 1861. Lorsque vous exécutez l'application, vous verrez un autre numéro de port.

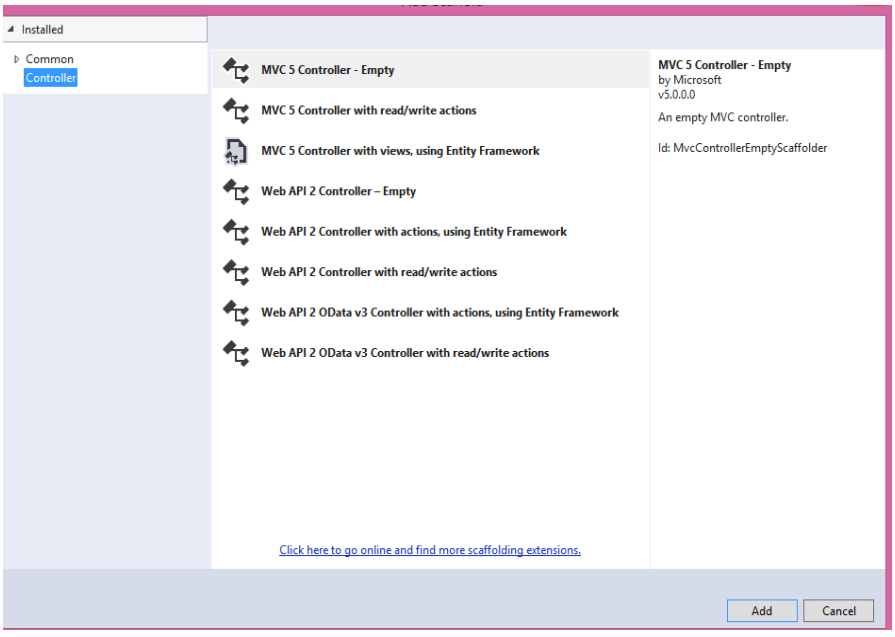


Création d'un contrôleur

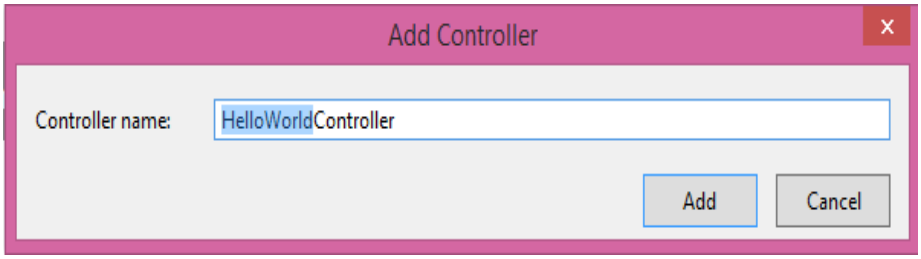
4. Nous allons commencer par créer une classe de contrôleur. Dans de **L'Explorateur de solutions**, faites un clic droit sur le dossier controllers, puis cliquez sur **Add**, puis **controller**.



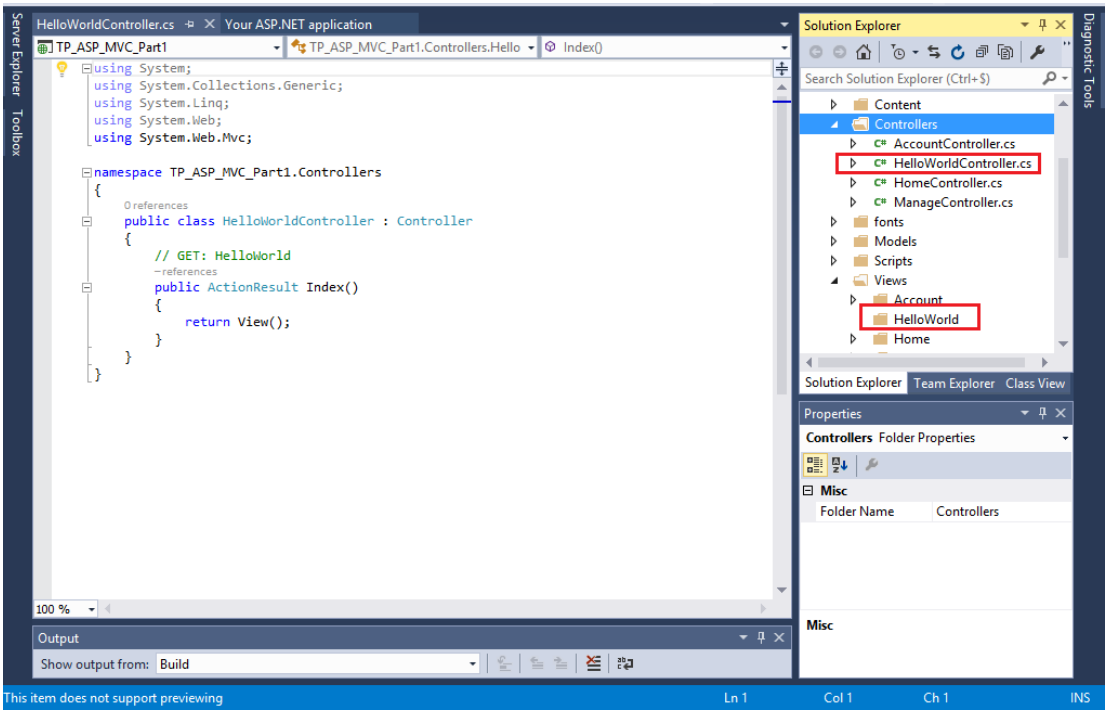
5. Puis cliquer sur **MVC 5 Controller – Empty ensuite** sur **Add**.



6. Nommez le nouveau contrôleur **HelloWorldController**



Dans **L'Explorateur** un nouveau fichier a été créé nommé ***HelloWorldController.cs*** et un nouveau dossier ***Views\HelloWorld***. Le contrôleur est ouvert dans l'IDE.

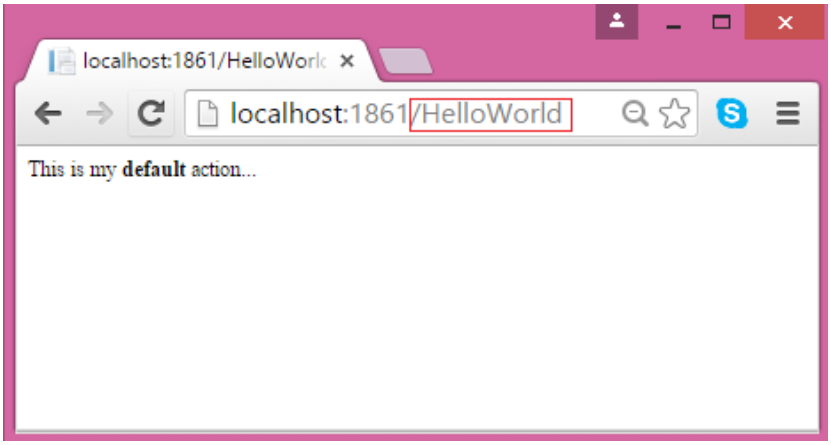


7. Remplacez le contenu du fichier avec le code suivant.

```
using System.Web;
using System.Web.Mvc;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        // GET: /HelloWorld/
        public string Index()
        {
            return "This is my <b>default</b> action...";
        }
        // GET: /HelloWorld/Welcome/
        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
    }
}
```

Le contrôleur est appelé **HelloWorldController** et la première méthode ou **Action** est nommée **Index**. Nous allons l'appeler à partir d'un navigateur.

8. Exécutez l'application (en appuyant sur F5 ou Ctrl + F5).Dans le navigateur, ajoutez «HelloWorld » au chemin d'accès dans la barre d'adresse.le code renvoie directement une chaîne.



Configuration des routes

ASP.NET MVC invoque les différents contrôleurs et leurs différentes actions selon l'URL entrante.
La logique de routage URL utilisée par défaut dans ASP.NET MVC utilise un format comme celui-ci pour déterminer quel code invoquer : **/[Controller]/[ActionName]/[Parameters]**

Vous définissez le format de routage dans le fichier **App_Start/RouteConfig.cs.**

Lorsque vous exécutez l'application et vous ne fournissez pas des segments d'URL, il utilise par défaut le contrôleur « Home » et la méthode d'action « Index » spécifié dans la section valeurs par défaut du code ci-dessous.

```
public static void RegisterRoutes(RouteCollection routes)
{
      routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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

- La première partie de l'URL détermine le contrôleur qu'on veut exécuter. Si ***/HelloWorld*** alors la classe **HelloWorldController** sera appelée.
- La deuxième partie de l'URL détermine la méthode d'action à Exécuter. Si ***/HelloWorld/Index*** alors la méthode Index de la classe HelloWorldController sera exécutée.

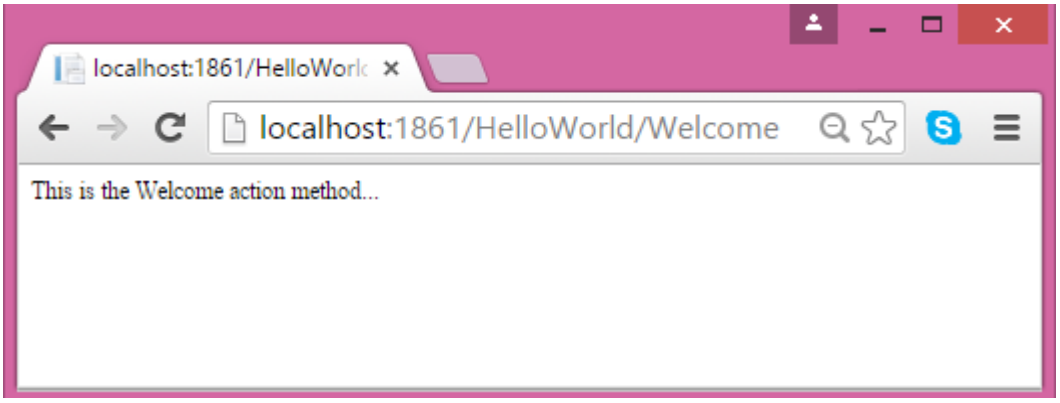
Une méthode nommée Index est la méthode par défaut et elle sera appelée sur un contrôleur si on n'a pas spécifié la méthode explicitement.

- La troisième partie du segment URL (Parameters) c'est pour acheminer les données.

Accédez à http://localhost:xxxx/HelloWorld/Welcome.

La méthode d'action Welcome s'exécute et retourne la chaîne "This is the Welcome action method..."

Le mappage de MVC par défaut est /[Controller]/[ActionName]/[Parameters]. Pour cette URL, le contrôleur est HelloWorld et Welcome est l'action. Nous n'avons pas utilisé les paramètres [Parameters] dans l'URL.



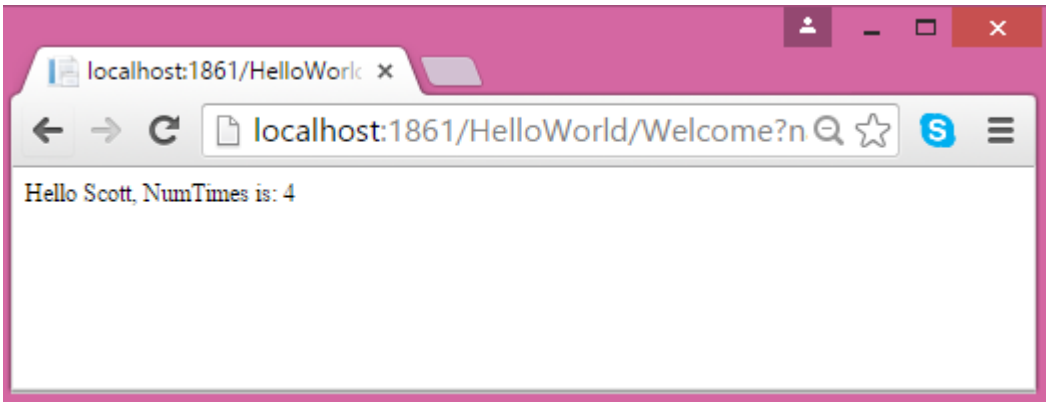
Nous allons modifier l'exemple un peu afin que vous pouvez passer des informations de paramètre de l'URL au contrôleur (par exemple, */HelloWorld/Bienvenue? nom = Scott & numtimes = 4*).

9. Changer votre méthode Welcome pour inclure deux paramètres comme indiqué ci-dessous. Notez que le code utilise la fonctionnalité de paramètre optionnel C# pour indiquer que le paramètre numTimes est par défaut égal à 1(si aucune valeur n'est passée pour ce paramètre).

```
public string Welcome(string name, int numTimes = 1 ) {
    return HttpUtility.HtmlEncode("Hello " + name + ", NumTimes is: " + numTimes);
}
```

Remarque de sécurité : Le code ci-dessus utilise HttpServerUtility.HtmlEncode pour protéger l'application des entrées malveillantes (à savoir le JavaScript). Lancez votre application et accédez à **l'URL de l'exemple (http://localhost:xxxx/HelloWorld/Welcome? name = Scott & numtimes = 4).**

Vous pouvez essayer différentes valeurs pour name et numtimes dans l'URL. Le système de liaison pour le modèle MVC ASP.NET mappe automatiquement les paramètres nommés de la chaîne de requête dans la barre d'adresse aux paramètres dans votre méthode.



Dans l'exemple ci-dessus, le segment d'URL (Parameters) n'est pas utilisé, les paramètres name et numTimes sont transmis sous forme de paramètres de requête. Le ? dans l'URL ci-dessus est un séparateur. Le & caractère sépare les paramètres de requête.

10. Remplacez la méthode d'action Welcome avec le code suivant :

```
public string Welcome(string name, int ID = 1)
{
    return HttpUtility.HtmlEncode("Hello " + name + ", ID: " + ID);
}
```

Cette fois le troisième segment de l'URL correspondant du paramètre route ID. La méthode d'action Welcome contient un paramètre (ID) correspondant à la spécification de l'URL dans la méthode RegisterRoutes.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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

11. Exécutez l'application et entrez l'adresse URL suivante : **http://localhost:xxxx/HelloWorld/Welcome/3?**

Dans les applications ASP.NET MVC, il est plus courant de passer des paramètres en tant que **paramètre de routes** (comme nous le faisions avec ID ci-dessus) que les transmettre sous forme de chaînes de requête. Vous pourriez également ajouter une route pour passer le name et numtimes dans les paramètres sous forme de données d'itinéraire dans l'URL. Dans le fichier *App_Start\RouteConfig.cs*, ajoutez la route de "Hello" :

```
public class RouteConfig
{
  public static void RegisterRoutes(RouteCollection routes)
  {
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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

    routes.MapRoute(
         name: "Hello",
         url: "{controller}/{action}/{name}/{id}" );
  }
}
```

12. Exécutez l'application et accédez à **/localhost:XXX/HelloWorld/Welcome/Scott/3.**

Vous pouvez spécifier des routes à l'aide de l'attribut Route de cette façon

```
[Route("WelcomeYou/{name}/{numTimes:int}")]
public string WelcomeYou(string name, int numTimes = 1)
{
        return HttpUtility.HtmlEncode("Hello " + name + ", NumTimes is: " + numTimes);
}
```

Ainsi vous pouvez accéder à cette action avec l'url : **/localhost:XXX/WelcomeYou/Sami/15.**

Remarquer l'absence du nom du contrôleur dans l'Url et la contrainte **int** sur le paramètre de route **numTimes**

Mais il faut activer ce mode de routage dans la méthode RegisterRoute :

```
public class RouteConfig
{
  public static void RegisterRoutes(RouteCollection routes)
  {
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapMvcAttributeRoutes();

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

    routes.MapRoute(
        name: "Hello",
        url: "{controller}/{action}/{name}/{id}" );
  }
}
```

13. Exécutez l'application et accédez à : **/localhost:XXX/WelcomeYou/Sami/15.**

Ajout d'une vue

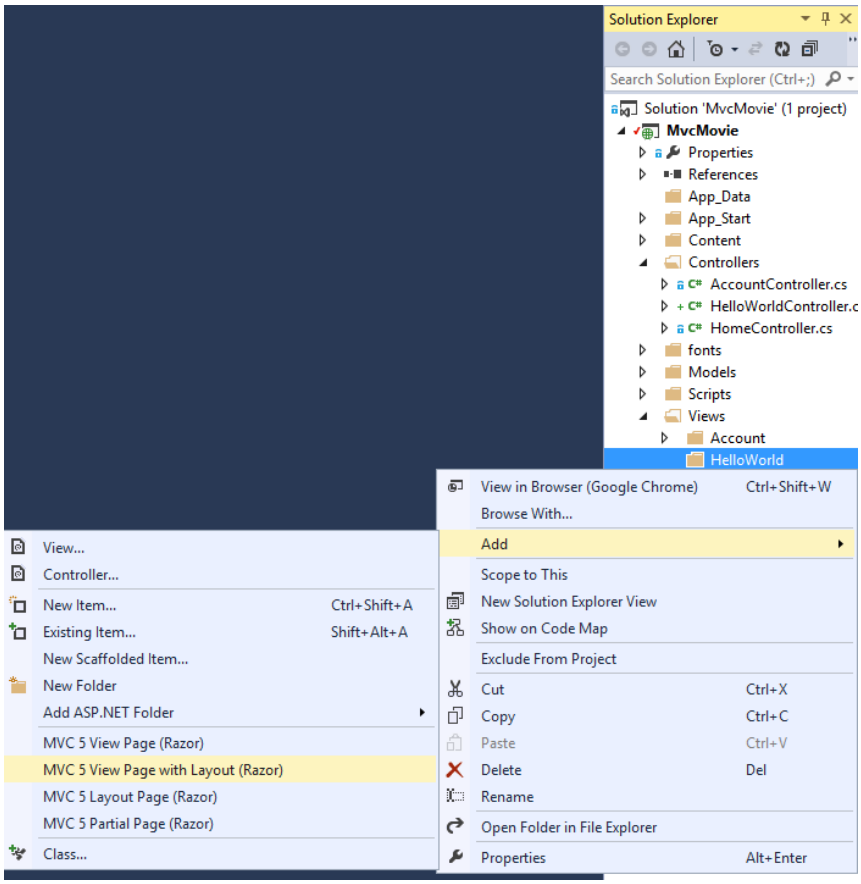
Dans cette section, vous allez modifier la classe `HelloWorldController` pour retourner des vues afin d'encapsuler le processus d'envois d'Html aux clients. Ces vues utiliseront le moteur d'affichage Razor et auront une extension de fichier *.cshtml* et fourniront un moyen élégant pour générer le HTML de sortie à l'aide de c#.

1. Actuellement, la méthode `Index` retourne une chaîne de caractères avec un message qui est codé en dur dans la classe de contrôleur. Changer la méthode `Index` pour renvoyer un objet `View` , comme illustré dans le code suivant :

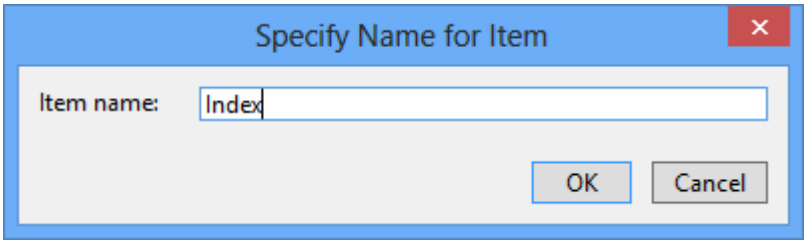
```
public ActionResult Index()
{
        return View();
}
```

La méthode de `Index` ci-dessus utilise un modèle d'affichage pour générer une réponse en HTML qui sera envoyer au navigateur du client. Les Méthodes des contrôleurs (également connues comme des action), telle que la méthode de `Index` ci-dessus, retournent généralement un ActionResult (ou une classe dérivée de ActionResult), et non pas des types primitifs comme les chaînes de caractères.

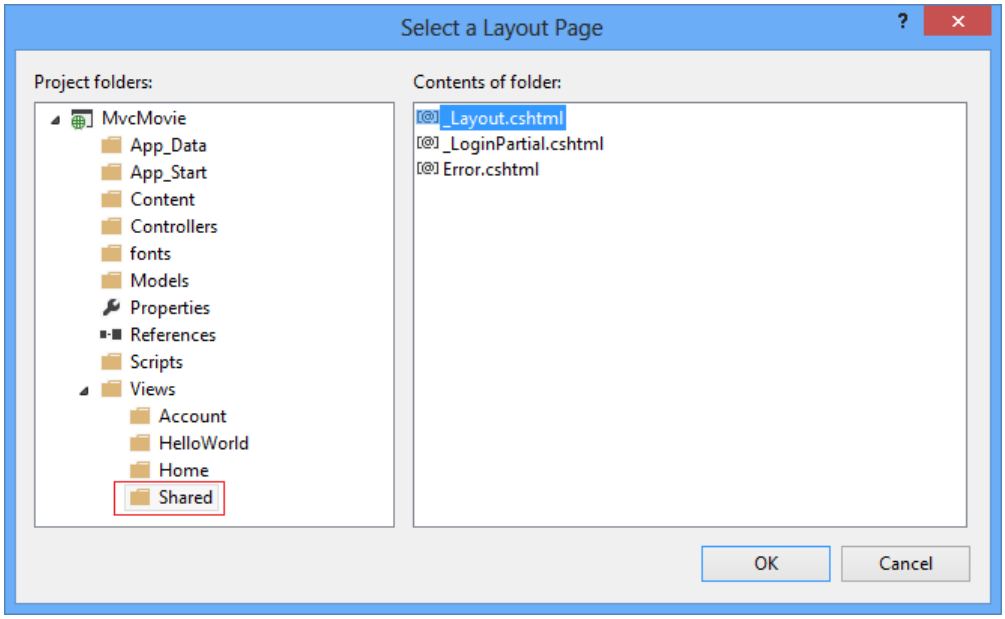
2. Clic droit sur le dossier *Views\HelloWorld* et cliquez sur **Ajouter**, puis cliquez sur **MVC 5 Voir Page avec (layout Razor)**.



3. Dans la boîte de dialogue **Spécifier le nom d'élément,** entrez *Index* puis cliquez sur **OK**.

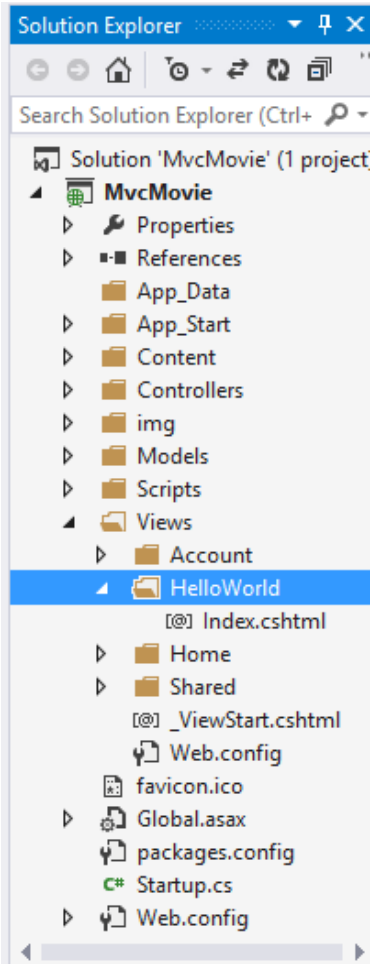


4. Dans la boîte de dialogue **Sélectionner une Page de disposition**, acceptez la valeur par défaut **_Layout.cshtml** , puis cliquez sur **OK**.



Dans la boîte de dialogue ci-dessus, le dossier *Views\Shared* est sélectionné dans le volet gauche. Si vous aviez un fichier de mise en page personnalisée dans un autre dossier, vous pouvez le sélectionner. Nous allons parler du fichier layout plus tard dans le tutoriel

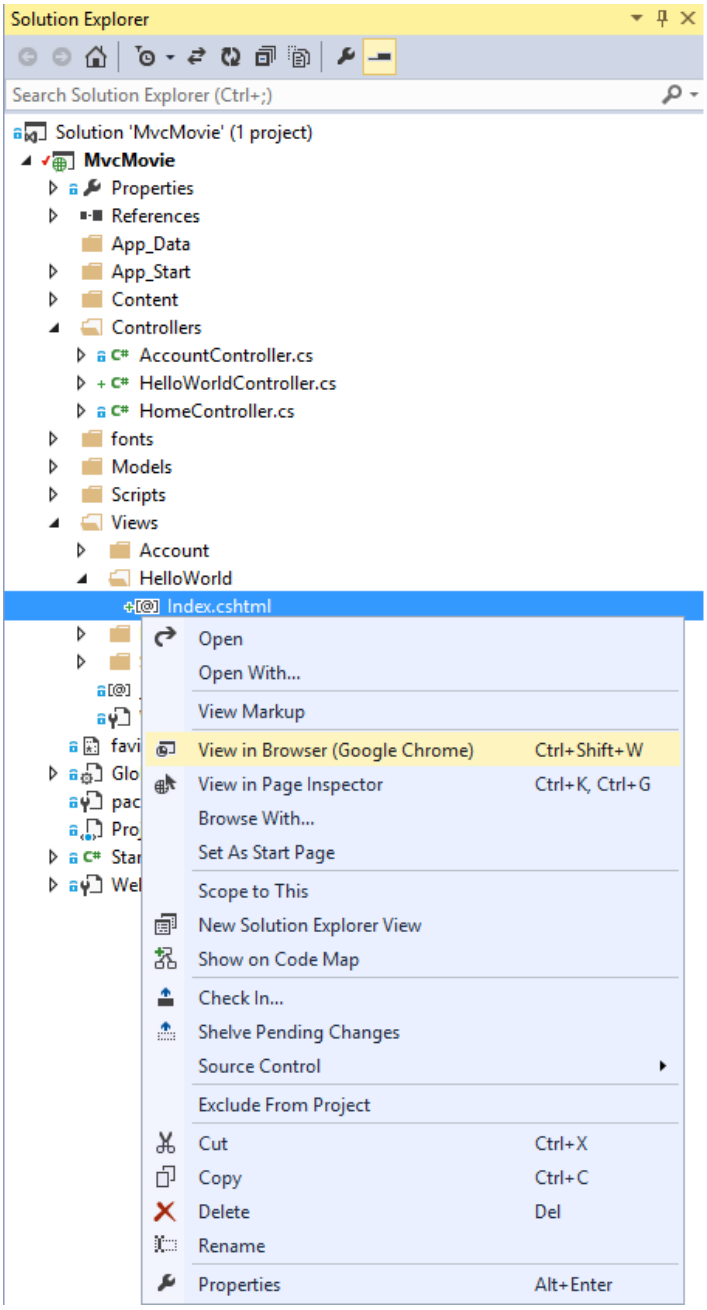
Le fichier *Views\HelloWorld\Index.cshtml* est créé.



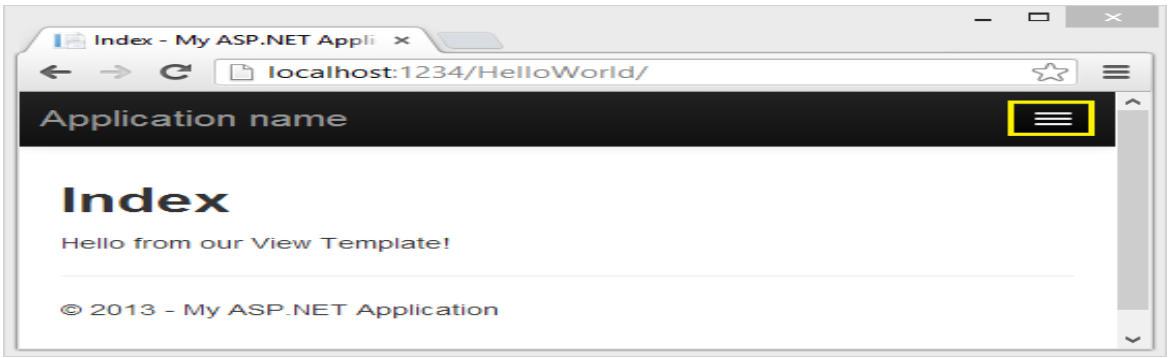
5. Ajoutez le code mis en évidence au fichier index.cshtml.

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

@{
      ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>Bonjour ceci est une vue!</p>
```

6. Faites un clic droit sur le fichier *Index.cshtml* et sélectionnez **afficher dans le navigateur**.



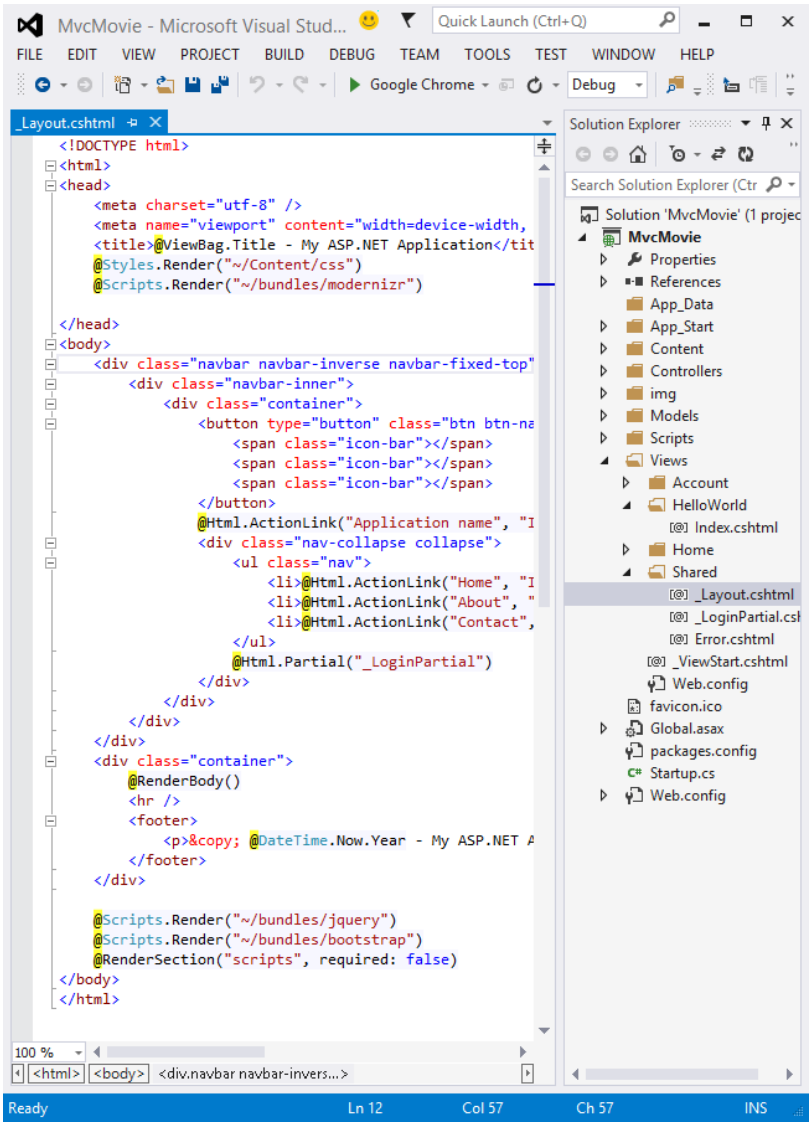
Vous pouvez également faire un click droit sur le fichier *Index.cshtml* et sélectionnez **voir en Page inspecteur.** Voir le tutoriel de l'inspecteur de la Page pour plus d'informations.



Personnalisation des vues et des Layout

Tout d'abord, on va modifier le lien « Nom de l'Application » en haut de la page. Ce texte est commun à chaque page. Il fait partie du Layout.

7. Allez dans le dossier */Views/Shared* dans **L'Explorateur de solutions** et ouvrez le fichier *_Layout.cshtml* . Ce fichier est le layout qui est utilisé par toutes les autres pages.



Le layout permet de préciser le Html du conteneur dans un seul endroit.

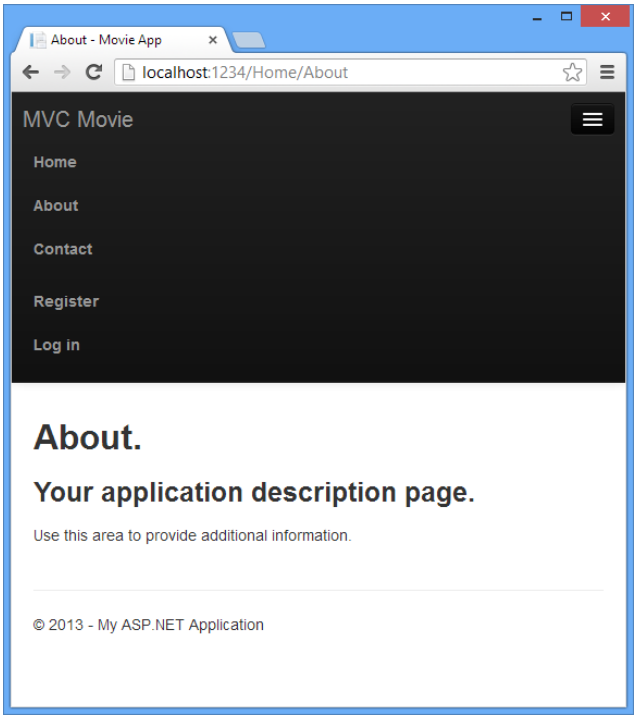
localiser `@RenderBody()`. `RenderBody` permet d'injecter les vues « encapsulées » dans le conteneur.

8. Modifier le contenu de l'élément title. Changer le texte de l' `ActionLink` dans le layout du "Nom de l'Application" à "MVC Movie" et le contrôleur de Home à Movies. Le fichier complet de mise en page est illustré ci-dessous :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - Movie APP</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("MVC Movie", "Index", "Movies", 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>
                @Html.Partial("_LoginPartial")
            </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>
```

9. Exécutez l'application et Notez le nouveau titre « MVC Movie ». Cliquez sur le lien About et vous voyez comment cette page affiche "MVC Movie". Nous avons été en mesure d'apporter la modification une fois dans le modèle de disposition et ont toutes les pages du site reflètent le nouveau titre.



Lorsque nous avons créé tout d'abord le fichier *Views\HelloWorld\Index.cshtml* , il contenait le code suivant :

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

Le code razor ci-dessus deifnit le layout. Examinez le fichier *Views_ViewStart.cshtml* , il contient la balise même exacte de Razor. Le fichier*Views_ViewStart.cshtml* définit le layout commun qui sera utilisé par toutes les vues, donc vous pouvez commenter ou supprimer ce code dans le fichier *Views\HelloWorld\Index.cshtml*.

```
@*@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}*@

@{
       ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>Bonjour ceci est une vue!</p>
```

Vous pouvez utiliser la propriété de `Layout` pour définir un layout différent ou aucun layout si elle est définie sur `null`.

Maintenant, nous allons changer le titre de la vue Index.

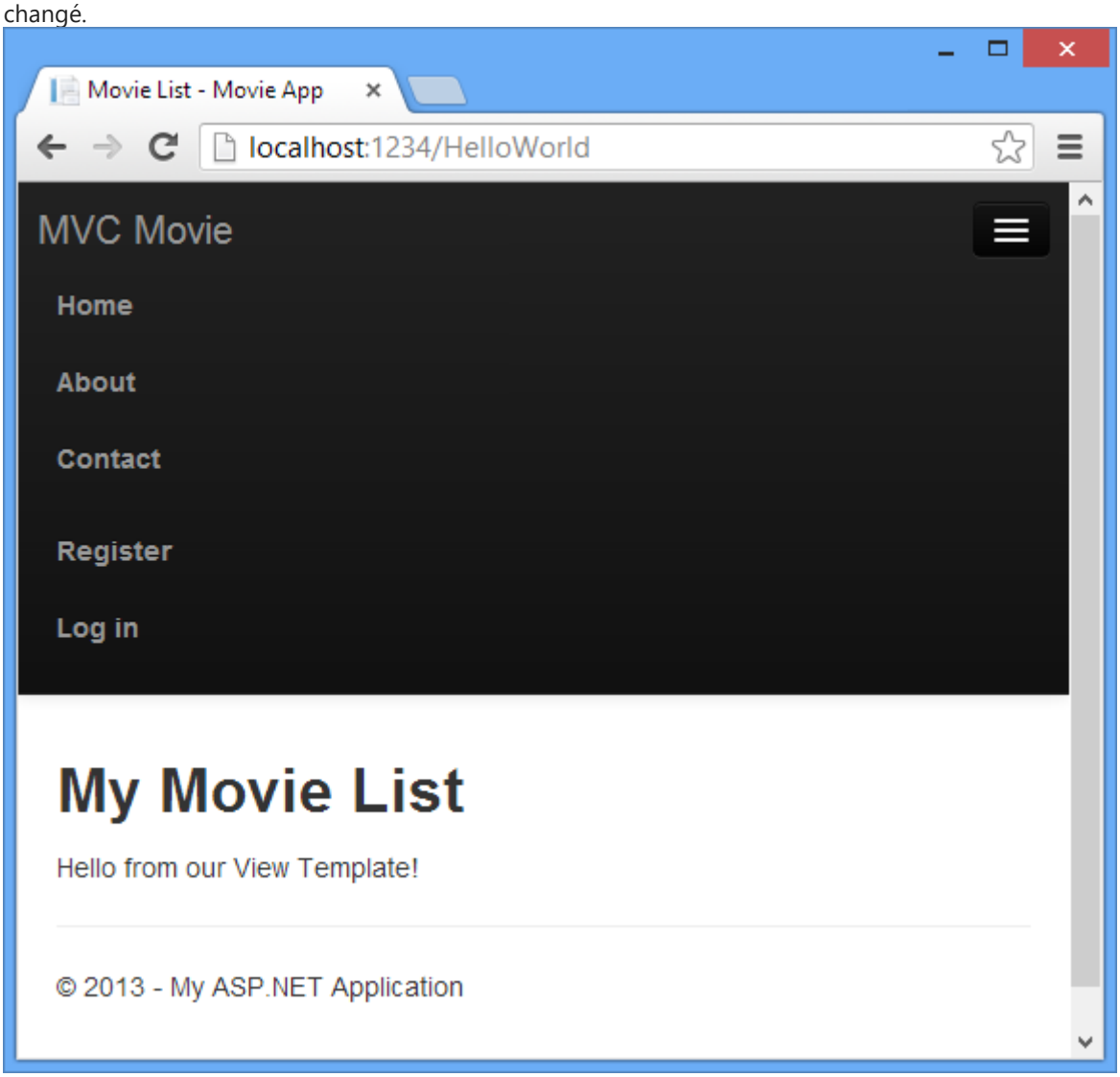
10. Ouvrez *MvcMovie\Views\HelloWorld\Index.cshtml*. Il y a deux endroits pour se faire un changement : tout d'abord, le texte qui apparaît dans le titre du navigateur, puis dans l'en-tête secondaire (l'élément `<h2>`)

```
@{
       ViewBag.Title = "Movie List";
}

<h2>My Movie List</h2>

<p>Hello from our View Template!</p>
```

Pour modifier le titre de la page qui est définit au niveau du layout on utilisera la propriété Title de l'objet `ViewBag`

En utilisant l'approche `ViewBag` , vous pouvez passer facilement des paramètres entre votre vue et layout.

11. Exécutez l'application. Notez que le titre du navigateur, le titre principal et les titres secondaires ont changé.



Pour l'instant, le message « Hello from notre modèle de vue! » est codé en dur, cependant. L'application MVC dispose d'un "V" (vue) et un "C" (contrôleur), mais pas "M" (modèle).

Passage de données du contrôleur à la vue

Les contrôleurs sont invoqués en réponse aux requêtes entrantes. Le controleur est la place où vous écrivez votre code qui fera appel à la couche métier, instancie des modèles et retourne des vues.

Les controlleurs sont responsables de fournir les données aux vues,

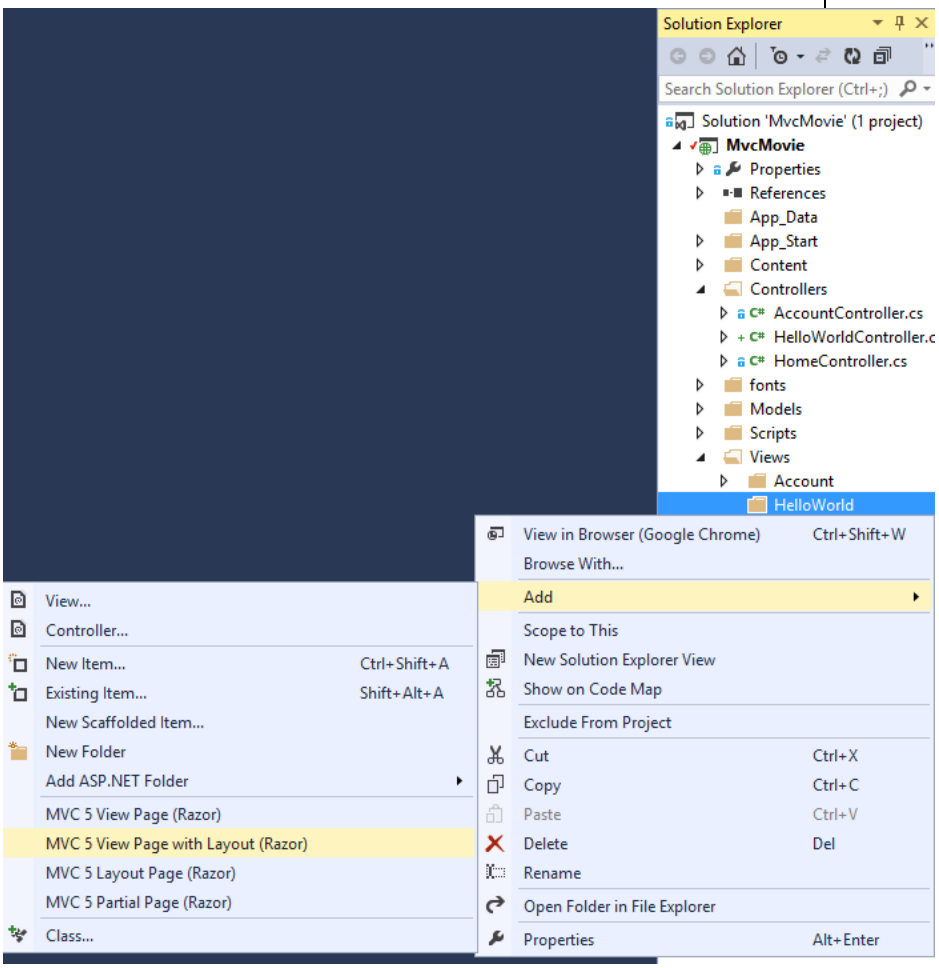
Actuellement, la méthode d'action `Welcome` dans la classe `HelloWorldController` prend un `name` et un paramètre `numTimes` et ensuite renvoie les valeurs directement dans le navigateur. Nous allons changer le contrôleur pour retourner une vue. Nous allons utiliser la propriété `ViewBag` pour passer les données du contrôleur à la vue.

12. Revenez au fichier *HelloWorldController.cs* et modifiez la méthode `Welcome` pour ajouter une valeur de `Message` et `NumTimes` à l'objet `ViewBag` . `ViewBag` est un objet dynamique, ce qui signifie que vous pouvez y mettre ce que vous voulez ; l'objet `ViewBag` n'a aucuns propriétés définis jusqu'à ce que vous mettre quelque chose dedans. Le système de liaison pour le modèle MVC ASP.NET mappe automatiquement les paramètres nommés (`name` et `numTimes`) de la requête dans la barre d'adresse aux paramètres dans votre méthode. Le fichier complet de *HelloWorldController.cs* ressemble à ceci :

```
using System.Web;
using System.Web.Mvc;

namespace MvcMovie.Controllers
{
        public class HelloWorldController : Controller
        {
                public ActionResult Index()
                {
                        return View();
                }

                public ActionResult Welcome(string name, int numTimes = 1)
                {
                        ViewBag.Message = "Hello " + name;
                        ViewBag.NumTimes = numTimes;

                        return View();
                }
        }


}
```

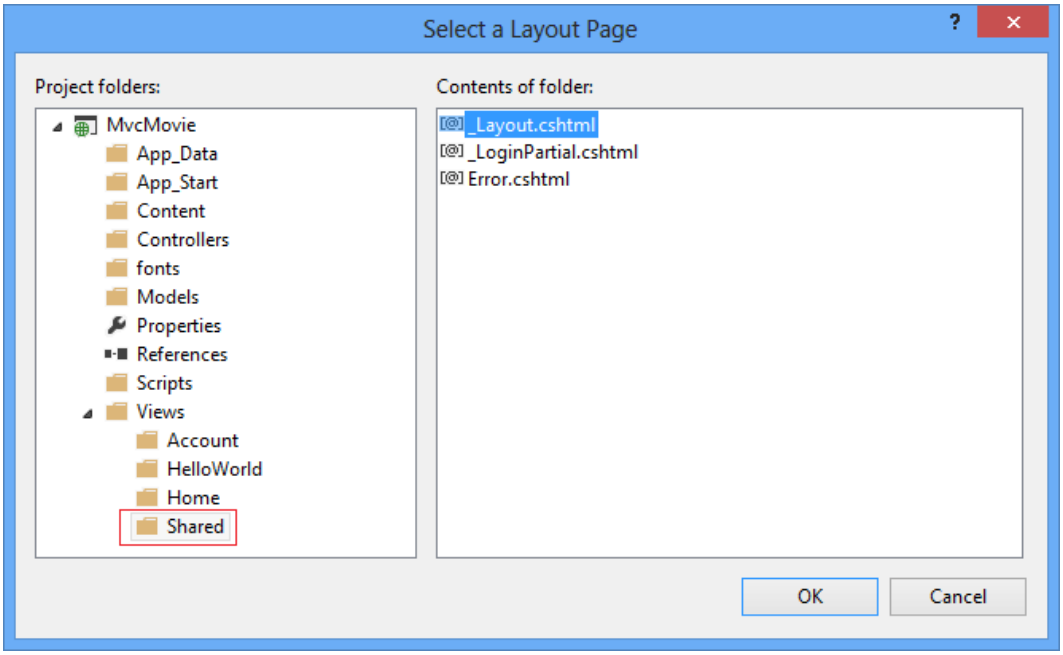
L'objet `ViewBag` contient maintenant des données qui seront transmises automatiquement à l'affichage. Ensuite, vous avez besoin d'une vue Welcome !

Dans le menu **Build** , sélectionnez **Build la Solution** (ou Ctrl + Maj + B) pour s'assurer que le projet est compilé. Clic droit sur le dossier *Views\HelloWorld* et cliquez sur **Ajouter**, puis cliquez sur **MVC 5 Voir Page avec (disposition Razor)**.



Dans la boîte de dialogue **Spécifier le nom d'élément** , entrez *Welcome* et puis cliquez sur **OK**.

Dans la boîte de dialogue **Sélectionner une Page de disposition** , acceptez la valeur par défaut **_Layout.cshtml** , puis cliquez sur **OK**.



Le fichier *MvcMovie\Views\HelloWorld\Welcome.cshtml* est créé.

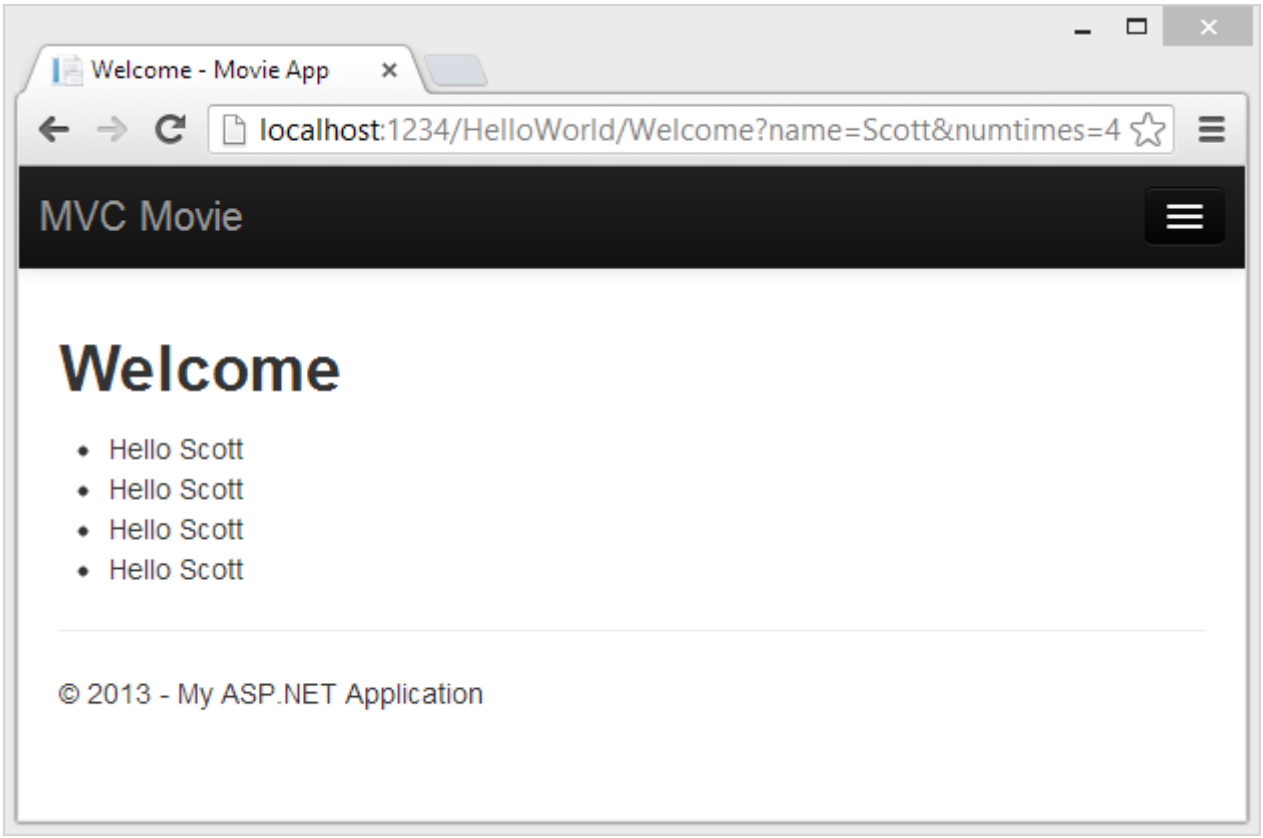
13. Remplacez la balise dans le fichier *Welcome.cshtml* . Vous allez créer une boucle qui dit « Hello » autant de

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

<h2>WelcomeView</h2>

<ul>
       @for (int i = 0; i < ViewBag.NumTimes; i++)
       {
       <li>@ViewBag.Message</li>
       }


</ul>
```

14. Exécutez l'application, puis accédez à l'URL suivante :

http://localhost:XX/HelloWorld/Welcome?name=Esprit&numtimes=4

Maintenant, les données sont tirées de l'URL et transmises au contrôleur en utilisant le model binder. Le contrôleur empaquète les données dans un objet `ViewBag` et passe cet objet à la vue. La vue affiche ensuite les données au format HTML à l'utilisateur.



Dans l'exemple ci-dessus, nous avons utilisé un objet `ViewBag` pour transmettre les données du contrôleur à une vue. Plus tard, nous allons utiliser un view modèle pour transférer des données d'un contrôleur à une vue. L'approche view modèle est généralement beaucoup préférable à l'approche ViewBag. Voir l'article de blog V fortement typées vues dynamiques pour plus d'informations.