

Styles et Layout

ASP.NET MVC

Styles et Layout

Razor apporte, avec les layouts, une solution afin de centraliser l'aspect de notre web application.

Le principe est de regrouper, dans une page spécifique, les éléments communs de mise en page pour le site (tels que le menu, l'entête, le pied de page, ou autres scripts JavaScript utilisés par toutes les pages).

Par défaut, lors de la création d'un site Web utilisant le moteur de vue Razor le fichier de `_Layout` est créé dans `/Views/Shared`

```
_Layout.cshtml*  X
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>@ViewBag.Title - My ASP.NET Application</title>
7   @Styles.Render("~/Content/css")
8   @Scripts.Render("~/bundles/modernizr")
9
10 </head>
11 <body>
12   <div class="navbar navbar-inverse navbar-fixed-top">
13     <div class="container">
14       <div class="navbar-header">
15         <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
16           <span class="icon-bar"></span>
17           <span class="icon-bar"></span>
18           <span class="icon-bar"></span>
19         </button>
20         @Html.ActionLink("Application name", "Index", "Home", new { area = "" },
21           new { @class = "navbar-brand" })
22       </div>
23       <div class="navbar-collapse collapse">
24         <ul class="nav navbar-nav">
25           <li>@Html.ActionLink("Home", "Index", "Home")</li>
26           <li>@Html.ActionLink("About", "About", "Home")</li>
27           <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
28         </ul>
29         @Html.Partial("_LoginPartial")
30       </div>
31     </div>
32   </div>
33   <div class="container body-content">
34     @RenderBody()
35     <hr />
36     <footer>
37       <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
38     </footer>
39   </div>
40
41   @Scripts.Render("~/bundles/jquery")
42   @Scripts.Render("~/bundles/bootstrap")
43   @RenderSection("scripts", required: false)
44 </body>
45 </html>
--
```

Styles et Layout

La vue *Layout* contient le DocType, Head et le body comme toute page classique HTML...

La seule différence et non la moindre est l'appel des méthodes *RenderBody()*, des *Renders* et de la création d'une *RenderSection*

- **RenderBody**

L'appel à cette méthode permet de définir l'emplacement où sera affiché le rendu des vues envoyées via les *Controller*

- **Renders**

- @Styles.Render(...) : permet de spécifier l'emplacement où seront insérés les styles présents dans le Bundle passé en paramètre.
- @Scripts.Render(...) : permet de spécifier l'emplacement où seront insérés les scripts présents dans le Bundle passé en paramètre.

- **RenderSection**

Un layout peut avoir plusieurs sections (header, footer, script, ...).

Afin de définir ces « zones » et permettre lors du rendu d'injecter du contenu dans notre Layout, nous utiliserons l'instruction @RenderSection.

Bundling et Minification

ASP.NET MVC

Bundling et Minification

Ces deux techniques sont utilisées pour optimiser le temps de chargement.

Elles permettent de diminuer le nombre de requêtes vers le server et de réduire la taille des ressources demandées (CSS et javascript)

Bundling : Permet de regrouper les fichiers dans un seul bundle et réduire considérablement les temps de chargement de la page.

Minification : La minification est l'ensemble des différentes optimisations de code js et/ou css. Cela passe par la suppression des espaces inutiles mais également d'autres techniques tels la suppression des commentaires, l'abréviation des variables, ...

Bundling et Minification

Le système de Bundle est configurable via le fichier `/App_Start/BundleConfig.cs`.

Afin de regrouper les scripts/css ensemble, il suffit de compléter les bundles existants ou d'en créer un nouveau :

```
bundles.Add(new  
StyleBundle("VirtualPath", ["cdn"]).Include("~/PathToResource"));
```

Ou

```
bundles.Add(new  
ScriptBundle("VirtualPath", ["cdn"]).Include("~/PathToResource"));
```

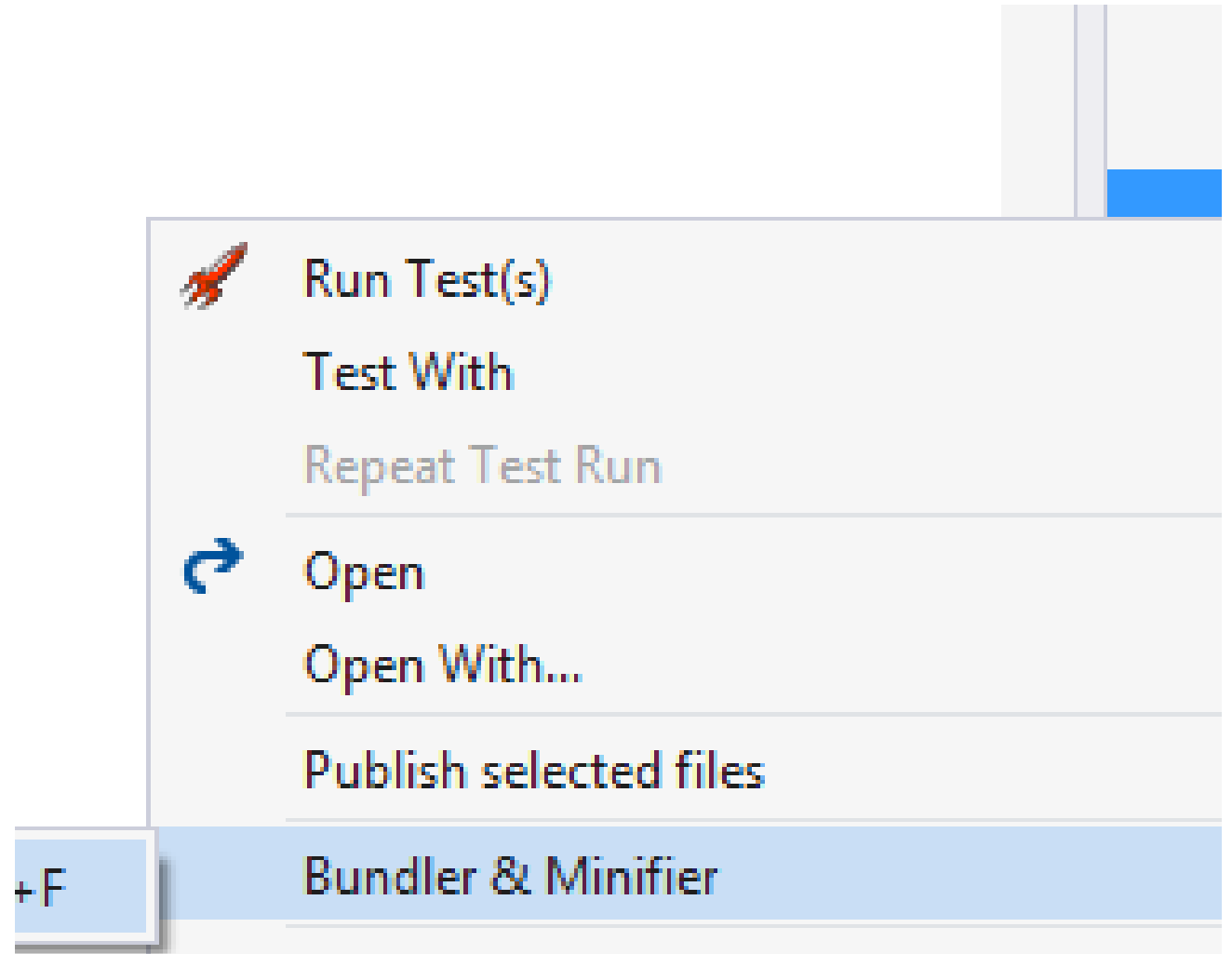
```
public class BundleConfig  
{  
    // For more information on bundling, visit http://go.microsoft.com/fwlink/?LinkId=301862  
    public static void RegisterBundles(BundleCollection bundles)  
    {  
        bundles.Add(new ScriptBundle("~/bundles/jquery").Include(  
            "~/Scripts/jquery-{version}.js"));  
  
        bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(  
            "~/Scripts/jquery.validate*"));  
  
        // Use the development version of Modernizr to develop with and learn from. Then, when you're  
        // ready for production, use the build tool at http://modernizr.com to pick only the tests you need.  
        bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(  
            "~/Scripts/modernizr-*"));  
  
        bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(  
            "~/Scripts/bootstrap.js",  
            "~/Scripts/respond.js"));  
  
        bundles.Add(new StyleBundle("~/Content/css").Include(  
            "~/Content/bootstrap.css",  
            "~/Content/site.css"));  
    }  
}
```

Bundling et Minification

Une extension visual studio (« Bundler & Minifier ») permet de faciliter la mise en place de la minification et du bundling.

Plus d'information sur

<https://visualstudiogallery.msdn.microsoft.com/9ec27da7-e24b-4d56-8064-fd7e88ac1c40>



Les contrôleurs

ASP.NET MVC

Les Contrôleurs

- **Le rôle du Contrôleur**

Il est chargé de répondre aux entrées utilisateur et , souvent, il doit également répercuter les changements sur le Modèle initiés par les entrées utilisateur.

En résumé, il doit donc se charger de toute communication entre le Modèle et la vue et générer le cas échéant les vues de réponse.

Dans le cas d'un site web statique, l'URL pointe directement vers le fichier HTML à produire.

Dans le cas d'un site web dynamique, l'URL pointe vers des scripts qui sont chargés de générer l'HTML à la volée.

En MVC, c'est légèrement différent :

L'URL indique au moteur de *Routing**, quel sera le contrôleur à instancier, quelle méthode à appeler dans celui-ci et fournit les arguments nécessaires à la méthode(*Action*)

L'action du contrôleur décide alors de la vue à utiliser et donc de l'HTML à générer

**Voir plus loin dans le cours*

Les Contrôleurs

Les bases d'un contrôleur

Nous allons parcourir les bases via une simple exemple : Le HomeController.

Ce HomeController est obtenu en choisissant le template de projet *ASP.NET Web Application* et en choisissant uniquement MVC.

Il est responsable de la Home Page, About Page et Contact Page.

```
public class HomeController : Controller Héríte de Controller
{
    public ActionResult Index() Home Page
    {
        return View();
    }

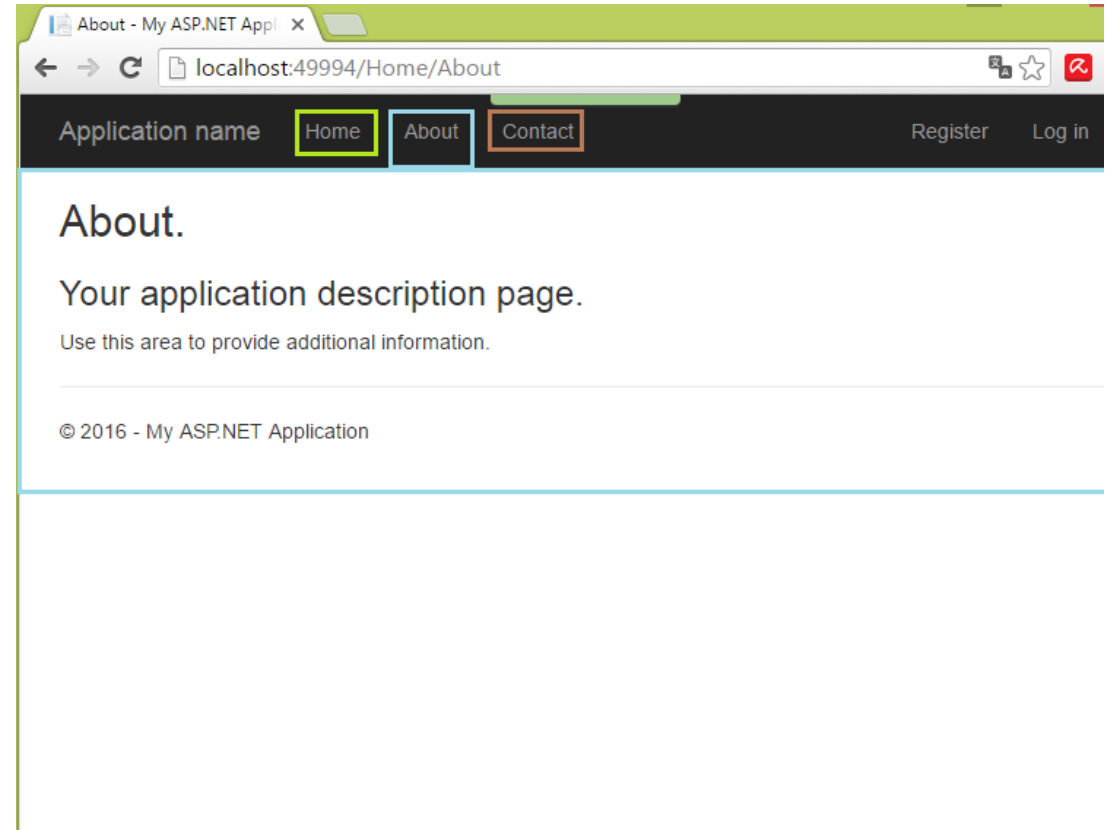
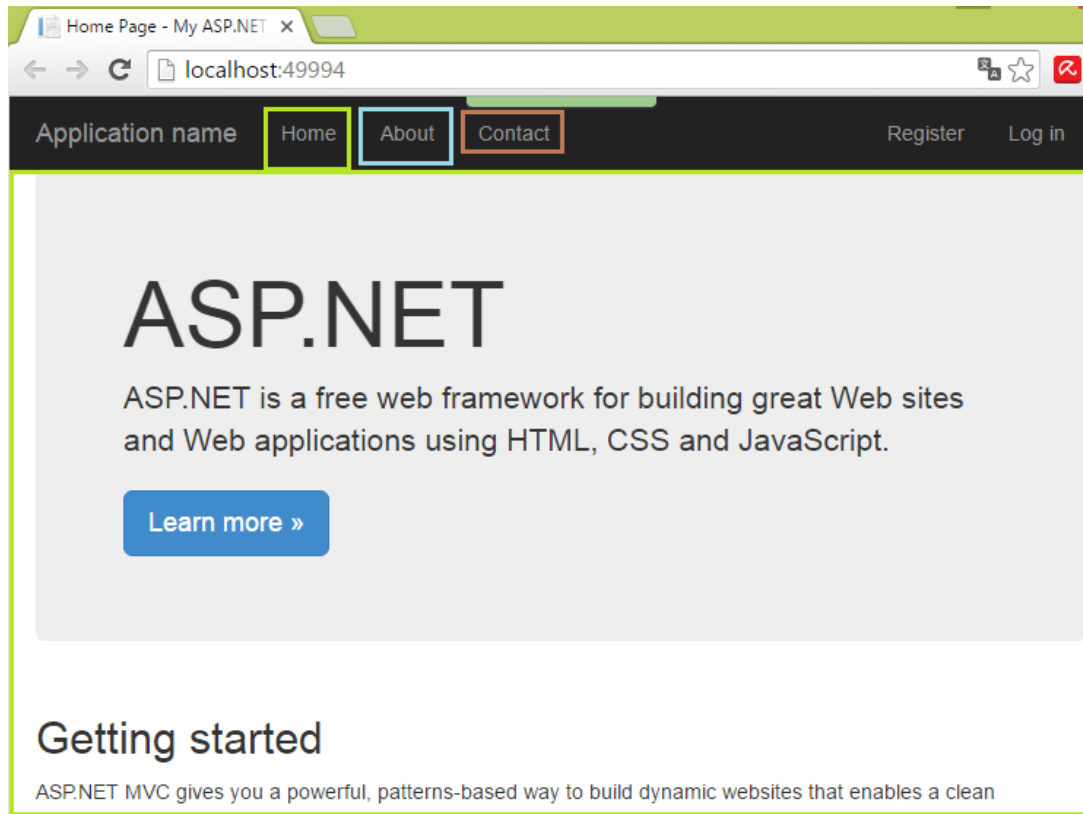
    public ActionResult About() About Page
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }

    public ActionResult Contact() Contact Page
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}
```

Les Contrôleurs



Les Contrôleurs

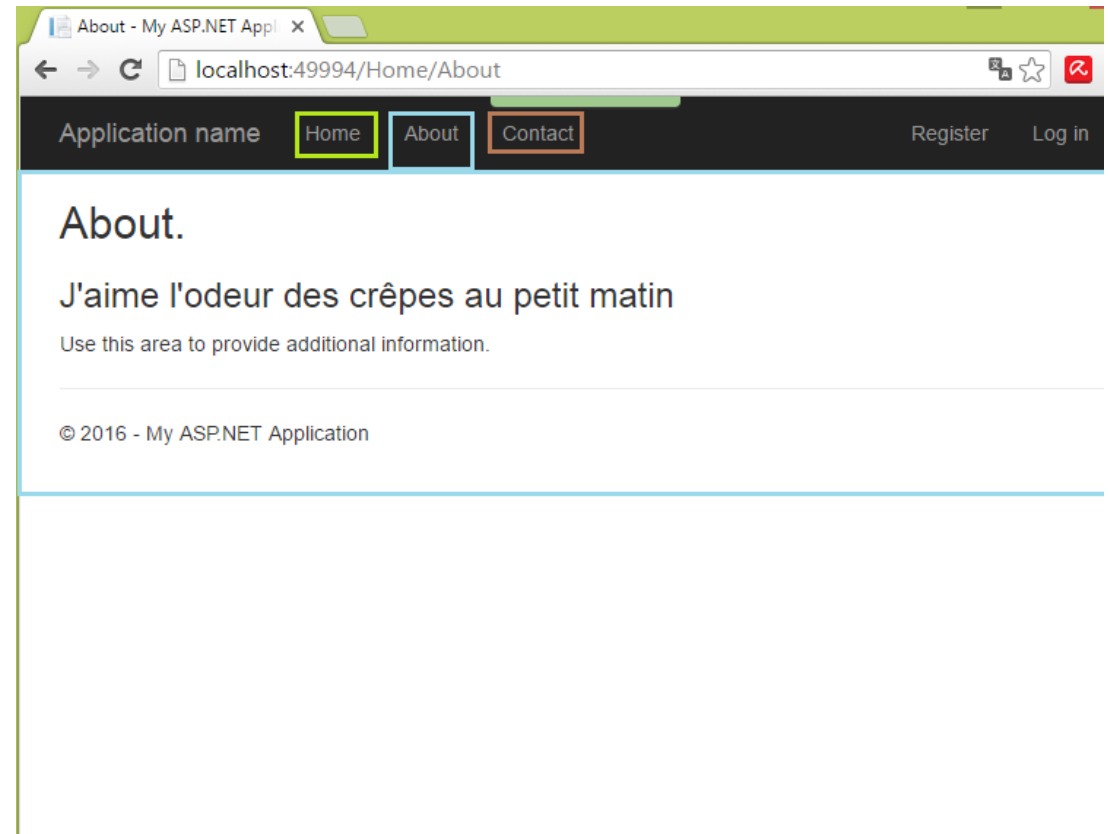
```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult About()
    {
        ViewBag.Message = "J'aime l'odeur des crêpes au petit matin";

        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

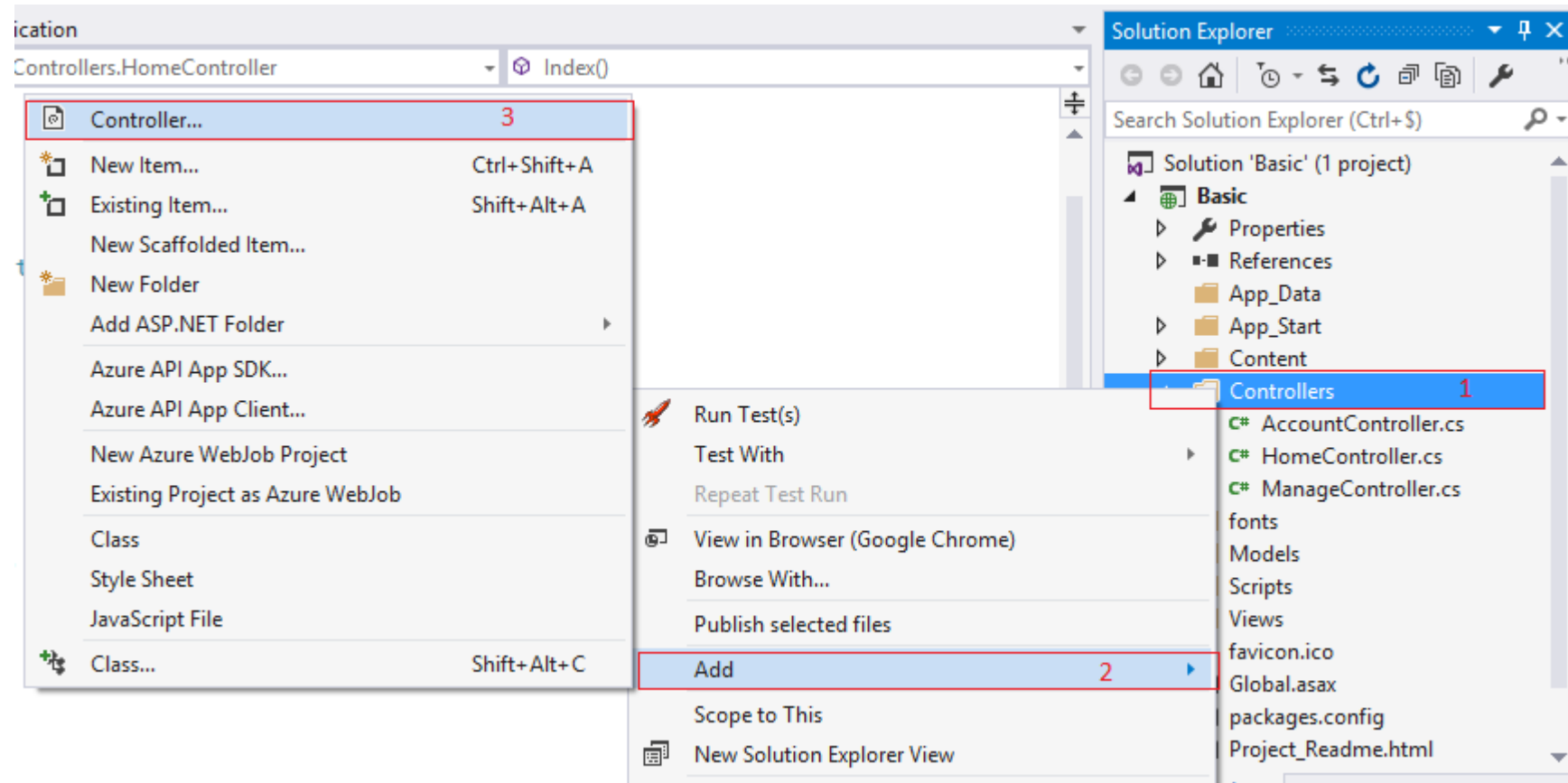
        return View();
    }
}
```



Les Contrôleurs

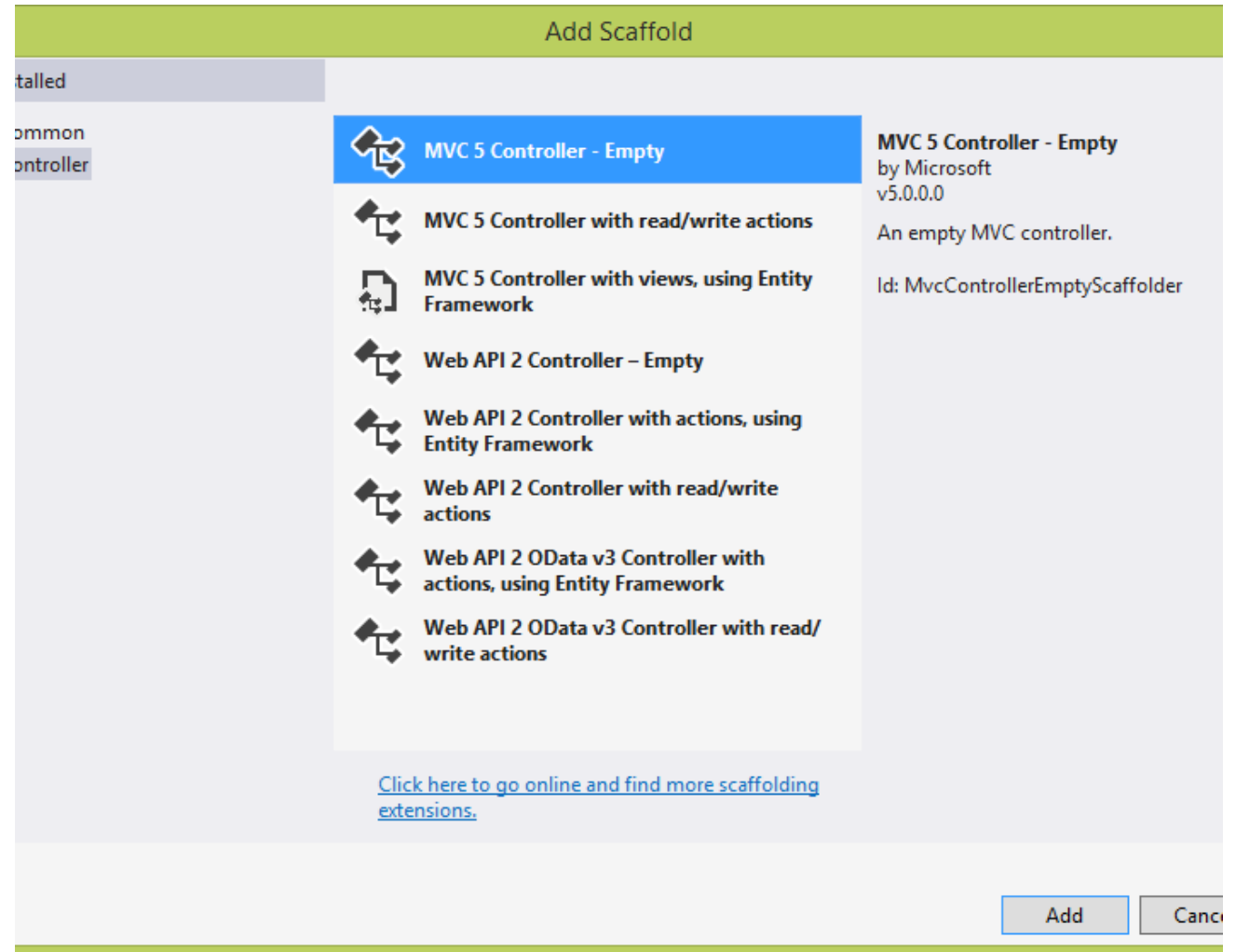
Comment créer un contrôleur

- 1- Click droit sur le dossier Controller
- 2- Add
- 3- Controller



Les Contrôleurs

4- Choisir le Empty scaffolding



Les Contrôleurs

5- Choisir un nom pour le controller

Remarque :

Le suffixe Controller est déjà encodé, veuillez à ne pas l'effacer

The diagram illustrates two sequential steps in adding a controller. The top form, titled 'Add Controller', features a text input field containing 'DefaultController' and an 'Add' button. A red arrow points from this form to a second, identical form below it. In the second form, the text input field contains 'StoreController', demonstrating the correct naming convention where the 'Controller' suffix is preserved.

Les Contrôleurs

Le contrôleur possède par défaut une action *Index*.

Nous pouvons changer le type de retour afin de par exemple renvoyer un String

```
namespace Basic.Controllers
{
    public class StoreController : Controller
    {
        // GET: Store
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



```
namespace Basic.Controllers
{
    public class StoreController : Controller
    {
        // GET: Store
        public string Index()
        {
            return "Nous somme sur Store.Index";
        }
    }
}
```


Les Contrôleurs

Nous pouvons également ajouter deux actions :

- Browse
- Details

Et ensuite lancer le projet et

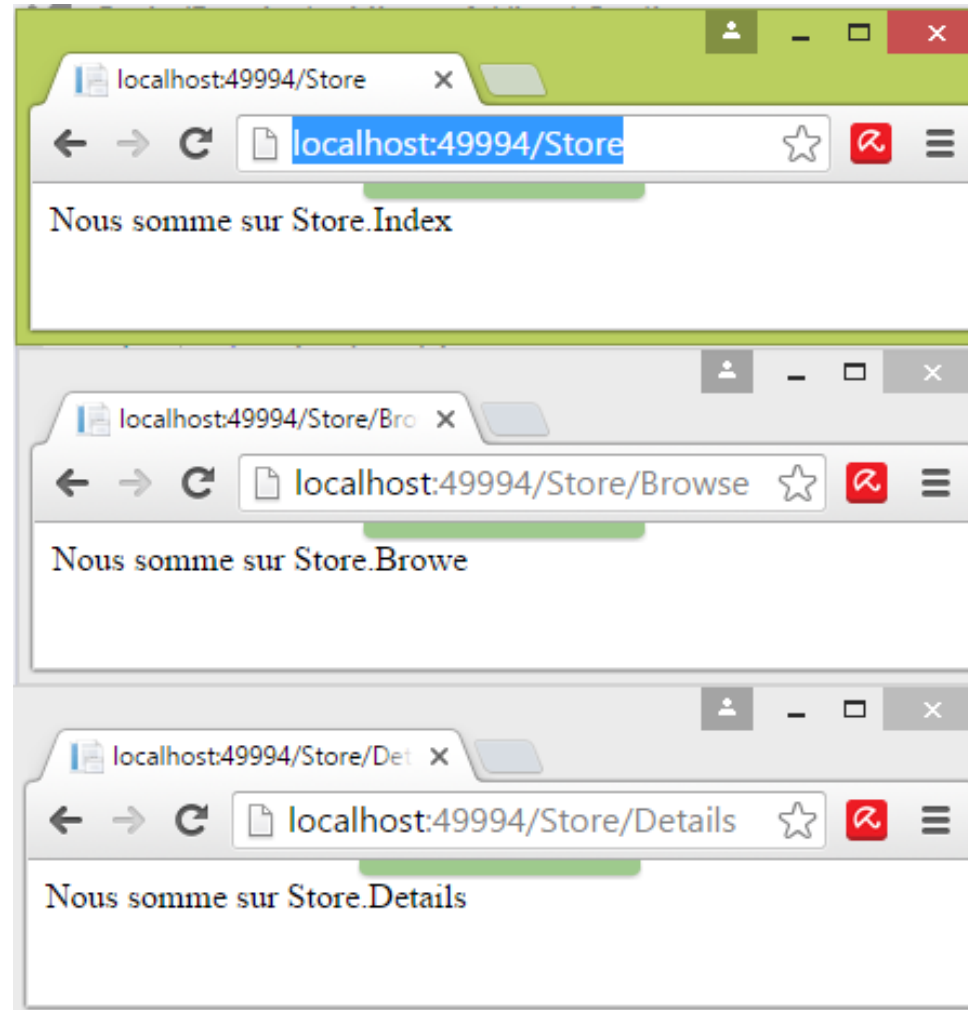
Naviguer vers les urls suivantes :

- /Store
- /Store/Browse
- /Store/Details

```
// GET: Store
public string Browse()
{
    return "Nous somme sur Store.Browse";
}

// GET: Store
public string Details()
{
    return "Nous somme sur Store.Details";
}
```

Les Contrôleurs



Les contrôleurs

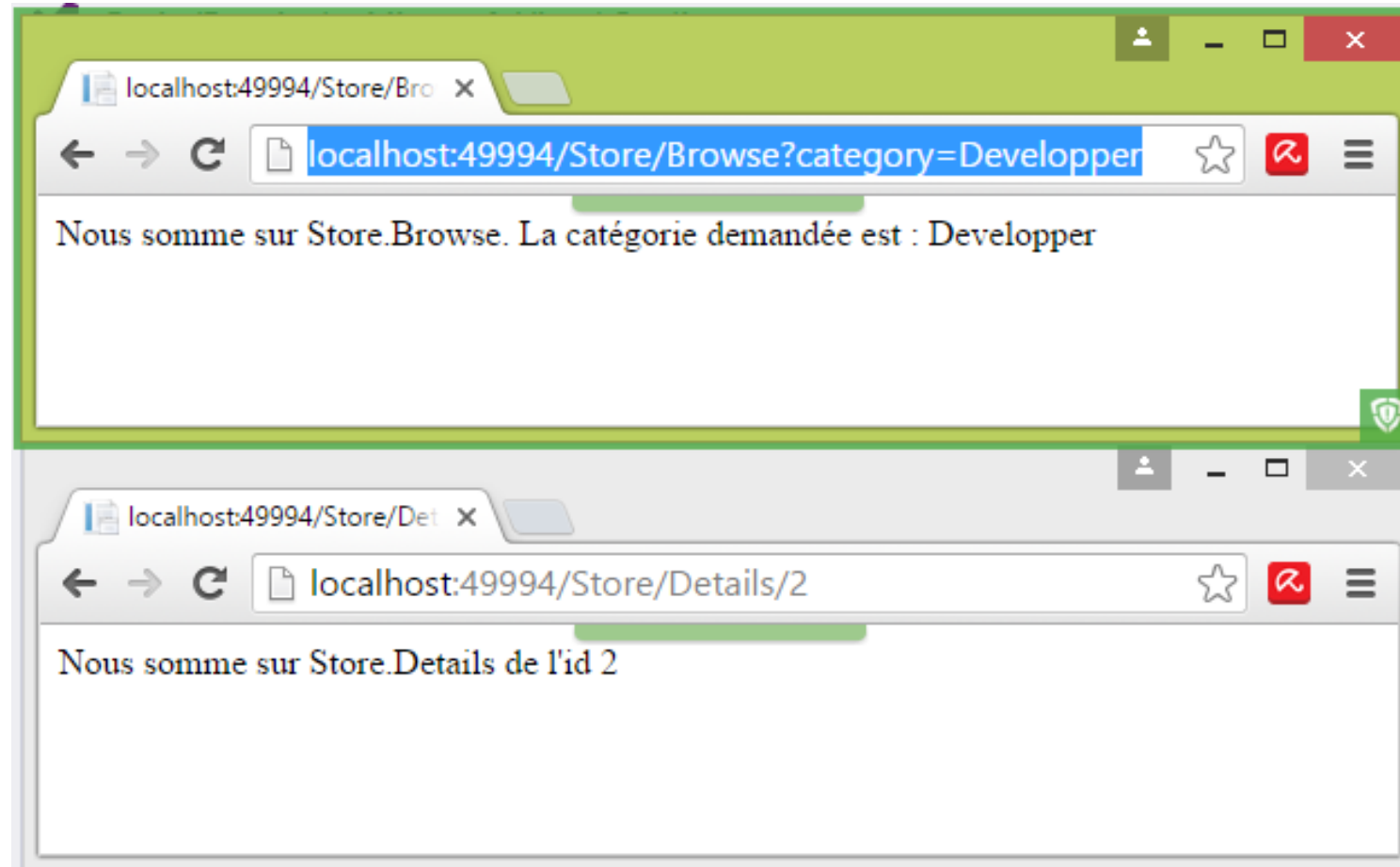
Comme toute méthode ou fonction, les Actions peuvent recevoir des paramètres.

Nous pouvons par exemple spécifier un paramètre *Category* pour l'action *Browse* et un paramètre *Id* pour l'action *Details*.

```
// GET: Store/Browse?category=Dev
public string Browse(string category)
{
    return @"Nous somme sur Store.Browse.
           La catégorie demandée est : "+ category;
}

// GET: Store/Details/5
public string Details(int id)
{
    return "Nous somme sur Store.Details de l'id "+ id;
}
```

Les contrôleurs



Les contrôleurs

- **La réception des données**

Un contrôleur a besoin d'accéder aux entrées de données (chaîne de caractères, formulaires, paramètres url,...)

- 1) **Extraire les données de l'objet Context**

Si on crée un contrôleur héritant de l'objet Controller, nous avons accès aux propriétés suivantes :

- Request
- Response
- RouteData
- HttpContext
- Server

Chacune de ces propriétés nous permet d'obtenir des informations par rapport à leur contexte d'exécution.

Les contrôleurs

Propriétés	Type	Description
Request.QueryString	NameValueCollection	Récupérer le GET
Request.Form	NameValueCollection	Récupérer le POST
Request.Cookies	HttpCookieCollection	Récupérer les cookies
Request.HttpMethod	String	Méthode utilisée (Get/POST)
Request.Headers	NameValueCollection	Le header HTTP complet
Request.Url	Uri	L'url
Request.UserHostAddress	String	L'adresse IP du client
RouteData.Route	RouteBase	L'entrée dans la table de routing de la ressource
RouteData.values	RouteValueDictionary	Les paramètres de la route
HttpContext.Application	HttpApplicationBase	La zone de stockage application
HttpContext.Cache	Cache	La zone de cache

Les contrôleurs

Propriétés	Type	Description
HttpContext.Item	IDictionary	Stockage de la request actuelle
HttpContext.Session	HttpSessionStateBase	Récupérer la Zone session
User	IPrincipal	Information de l'utilisateur logué
TempData	TempDataDictionary	Espace de stockage temporaire pour l'utilisateur courant

```
public ActionResult RenameProduct()
{
    // Accès aux propriétés de contexte
    string userName = User.Identity.Name;
    string serverName = Server.MachineName;
    string clientIP = Request.UserHostAddress;
    DateTime dateTimeStamp = HttpContext.Timestamp;
    string oldProductName = Request.Form["Nom"];
    string newProductName = Request.Form["Prenom"];
    return View("ProductRenamed");
}
```

Les contrôleurs

2) Utiliser les paramètres de la méthode Action

```
public ActionResult ShowWeatherForecast()  
{  
    string city = RouteData.Values["city"].ToString();  
    DateTime forDate = DateTime.Parse(Request.Form["forDate"]);  
}
```



```
public ActionResult ShowWeatherForecast(string city, DateTime forDate)  
{  
}
```

Nous n'avons rien d'autre à faire : le framework MVC va chercher les données dans les objets du contexte.

Remarque : Action n'autorise pas de passage via *Ref* ou *Out*

Les contrôleurs

La classe de Base *Controller* obtient les valeurs pour les paramètres en utilisant les composants du framework MVC : le *value provider* et le *model Binder*.

A. *Value provider*

Il représente l'ensemble des données disponibles. Il va rechercher les informations à partir de *Request.Form*, *Request.QueryString*, *Request.Files* et *RouteData.Value*.

B. *Model Binder*

Le model Binder par défaut peut créer et peupler n'importe quel type .NET.

Nous en reparlerons plus tard.

Si le framework ne peut pas trouver de valeurs pour un paramètre, celui-ci envoie un null ➔ Exception

Pour éviter cette exception, nous pouvons utiliser des types nullable (int?, string?,...) ou encore des paramètres avec valeurs par défaut (string query=« default »,...)

Les contrôleurs

L'envoi des données

Un Controller , après avoir traité la requête, doit généralement produire une réponse.

Pour cela, au lieu de directement faire appel à l'objet *Response* (*Response.Write*, *Response.redirect*,....) nous renvoyons un objet dérivé de *ActionResult*.

Quand le framework reçoit un objet *Actionresult*, il appelle la méthode *ExecuteResult* de la classe *Controller*.

```
public override void ExecuteResult(ControllerContext context)
{
    string destinationUrl = UrlHelper.GenerateContentUrl(Url, context.HttpContext);
    if (Permanent)
    {
        context.HttpContext.Response.RedirectPermanent(destinationUrl,
            endResponse: false);
    }
    else
    {
        context.HttpContext.Response.Redirect(destinationUrl, endResponse: false);
    }
}
```

Les contrôleurs

```
public class DerivedController : Controller
{
    //
    // GET: /Derived/

    public void Index()
    {
        string controller = (string)RouteData.Values["controller"];
        string action = (string)RouteData.Values["action"];
        Response.Write(string.Format("Controller: {0}, Action: {1}", controller, action));
    }

    public ActionResult Redirect()
    {
        return new RedirectResult("/Derived/Index");
    }
}
```

Les contrôleurs

Type	Controller helper
ViewResult	View
PartialViewResult	PartialView
RedirectToRouteResult	RedirectToAction, RedirectToActionPermanent, RedirectToRoute RedirectToRoutePermanent
RedirectResult	Redirect RedirectPermanent
ContentResult	Content
FileResult	File
JsonResult	Json
JavaScriptResult	JavaScript
HttpUnauthorizedResult	None
HttpNotFoundResult	HttpNotFound
HttpStatusCodeResult	None
EmptyResult	None

Les Vues

ASP.NET MVC

Les vues

- **But**

La vue est responsable de fournir l'Interface Utilisateur (UI).

En bref, lorsque le Contrôleur a terminé l'exécution de la logique business, il délègue l'affichage à la vue.

Remarque :

Contrairement aux autres framework basé sur les fichiers, les vues ne sont pas directement accessibles.

Nous ne pouvons pas préciser une URL pour atteindre directement la vue.

La vues est TOUJOURS « rendue » via un contrôleur qui fournit les données pour l'affichage si nécessaire.

Les vues

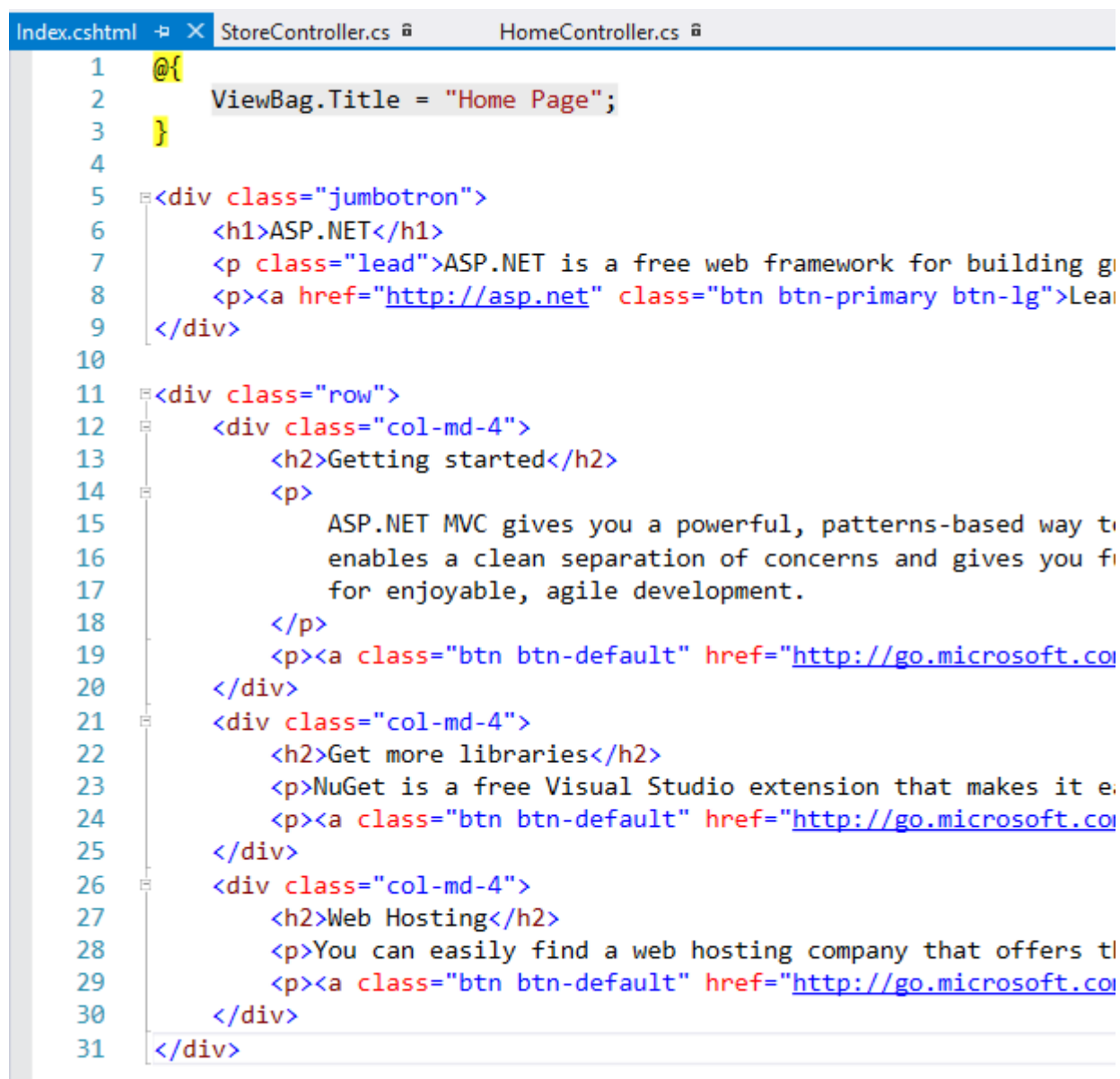
Les bases d'une vue

La vue la plus simple disponible, est celle qui n'a pas besoin d'informations (données) venant du contrôleur.

Exemple : Index.cshtml

Le rendu de cette vue se fait dans le contrôleur *Home*, dans l'action *Index*

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



```
1 @{
2     ViewBag.Title = "Home Page";
3 }
4
5 <div class="jumbotron">
6     <h1>ASP.NET</h1>
7     <p class="lead">ASP.NET is a free web framework for building g
8     <p><a href="http://asp.net" class="btn btn-primary btn-lg">Lea
9 </div>
10
11 <div class="row">
12     <div class="col-md-4">
13         <h2>Getting started</h2>
14         <p>
15             ASP.NET MVC gives you a powerful, patterns-based way to
16             enables a clean separation of concerns and gives you fi
17             for enjoyable, agile development.
18         </p>
19         <p><a class="btn btn-default" href="http://go.microsoft.co
20     </div>
21     <div class="col-md-4">
22         <h2>Get more libraries</h2>
23         <p>NuGet is a free Visual Studio extension that makes it e
24         <p><a class="btn btn-default" href="http://go.microsoft.co
25     </div>
26     <div class="col-md-4">
27         <h2>Web Hosting</h2>
28         <p>You can easily find a web hosting company that offers ti
29         <p><a class="btn btn-default" href="http://go.microsoft.co
30     </div>
31 </div>
```

Les vues

Cet exemple basique nous montre l'utilisation du mot clé *ViewBag*.

Le *ViewBag* permet de définir des propriétés ou des objets qui seront disponibles sur la vue.

Ces objets dynamiques sont accessible via *Controller.ViewBag*

```
public ActionResult Index()  
{  
    ViewBag.Message = "Hello";  
    ViewBag.Date = DateTime.Now;  
    return View();  
}
```



```
<div>  
    Nous sommes le @ViewBag.Date.DayOfWeek  
    Le message est @ViewBag.Message  
</div>
```

Remarque :

Le *ViewBag* est utilisée pour le passage de petites données.

En aucun cas, il ne faut utiliser le *ViewBag* pour passer des objets complexes.

Les vues

- **Les conventions**

Dans le contrôleur, nous avons simplement l'instruction `return View()`

Pas besoin de donner le nom de la vue car, par convention, le moteur MVC utilise la logique suivante :

→ Dossier View → Dossier portant le nom du contrôleur → Fichier portant le nom de l'action → Extension cshtml ou aspx ou asmx.

Les deux dernière extension sont moins souvent utilisées.

.chhtml = razor view

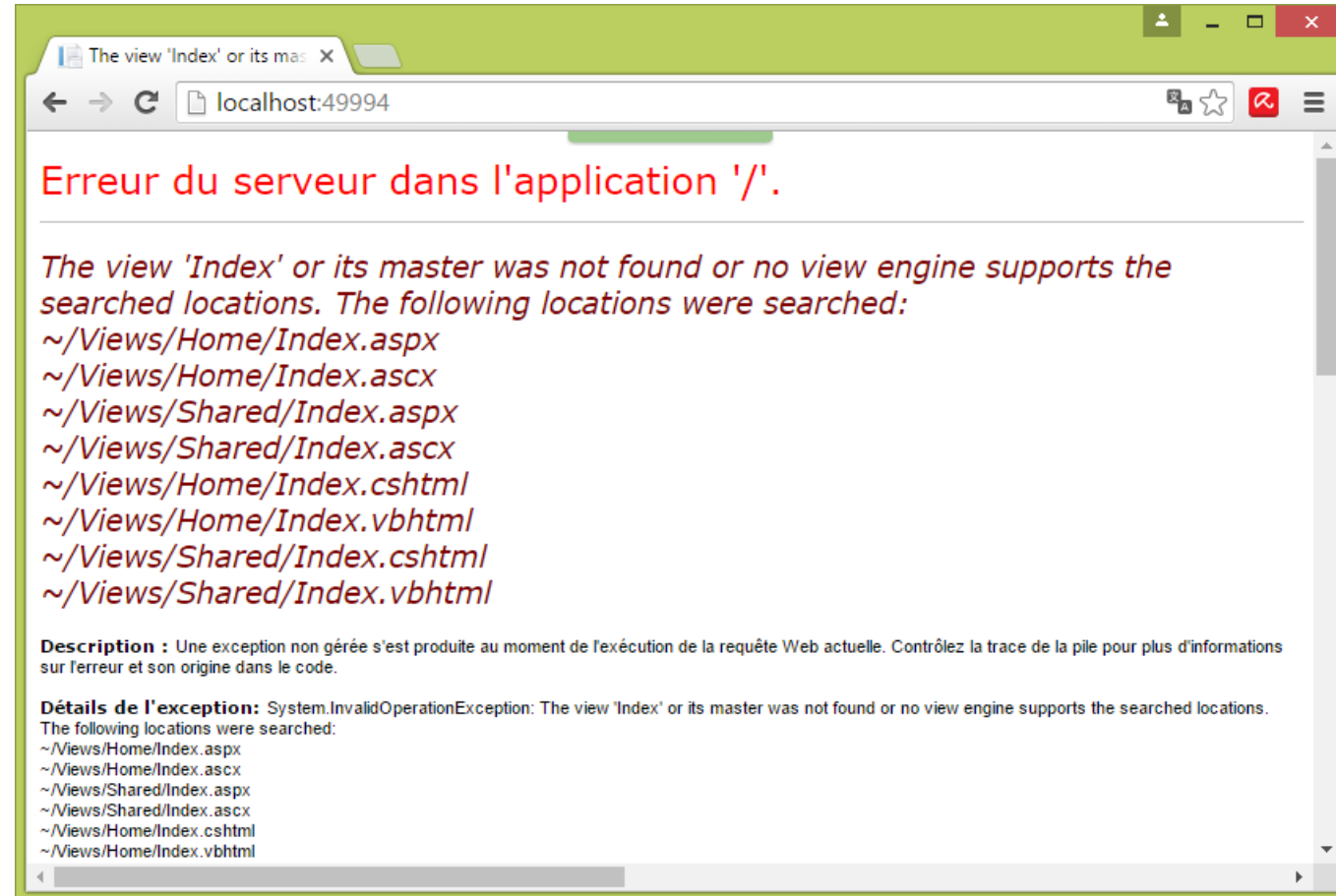
.aspx = WebForm (Old school)

.asmx = fichier webservice asp

Si le moteur MVC ne trouve pas de fichier approprié via la logique précédente, il regarde dans le dossier *View/Shared*.

Et si toujours pas de fichier correct, nous obtenons une exception.

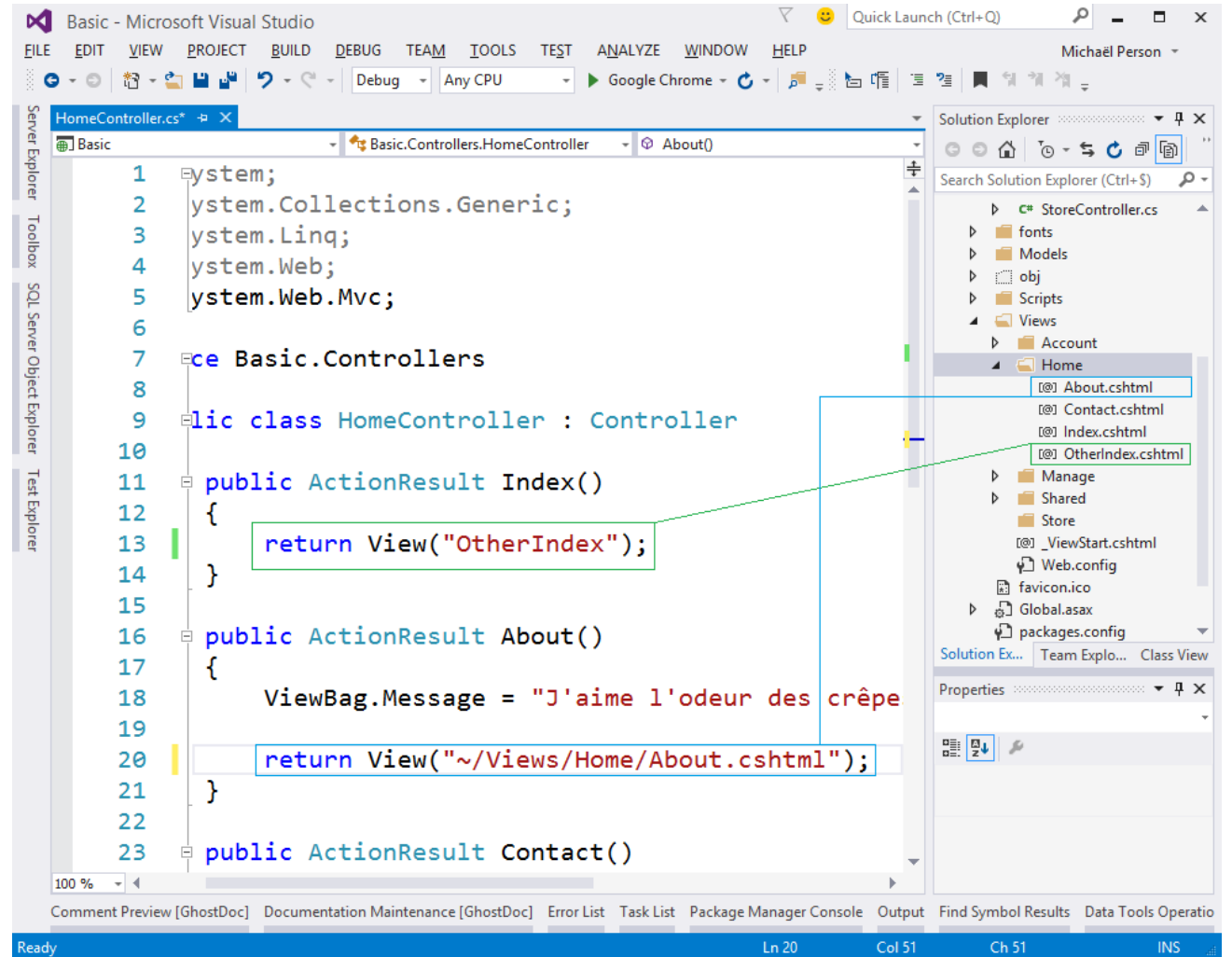
Les vues



Les vues

Il est cependant possible de contourner la convention.

- passant le nom de la vue
- En donnant le chemin complet vers la vue



Les vues

Le « ViewBag »

Les données gérées par les vues ne sont pas toujours en lien directe avec le model de domaine.

Une vue pourrait, par exemple, afficher les informations d'un produit mais également les informations de l'utilisateur courant, les catégories liées, etc...

Une façon simple, sans utiliser les modèles et les vues fortement typées que nous verrons plus tard, est d'utiliser le *ViewBag*.

Bien entendu, nous utiliserons cette technique pour uniquement des données « basique » et non pour des données complexes.

```
troller.cs -> X
Basic.Controllers.HomeController
Index()

6
7 namespace Basic.Controllers
8
9 public class HomeController : Controller
10
11 public ActionResult Index()
12 {
13     ViewBag.InfoUser = new { Nom = "Person", Prenom = "Mike" };
14     ViewBag.InfoProduit = new { Nom = "Eau", Prix = 2.14 };
15     return View("OtherIndex");
16 }
17

cshtml -> X
1
2 @{}
3     ViewBag.Title = "OtherIndex";
4 }
5
6 <h2>OtherIndex</h2>
7 <h3>Utilisateur</h3>
8 @ViewBag.InfoUser.Nom @ViewBag.ViewBag.InfoUser.Prenom
9 <hr />
10 @ViewBag.InfoProduit.Nom : @ViewBag.ViewBag.InfoProduit.Prix &euro;
11
12
```

Les vues

Créer une vue

La création d'une vue est grandement facilitée grâce à Visual Studio.

La façon la plus simple est de partir de l'action :

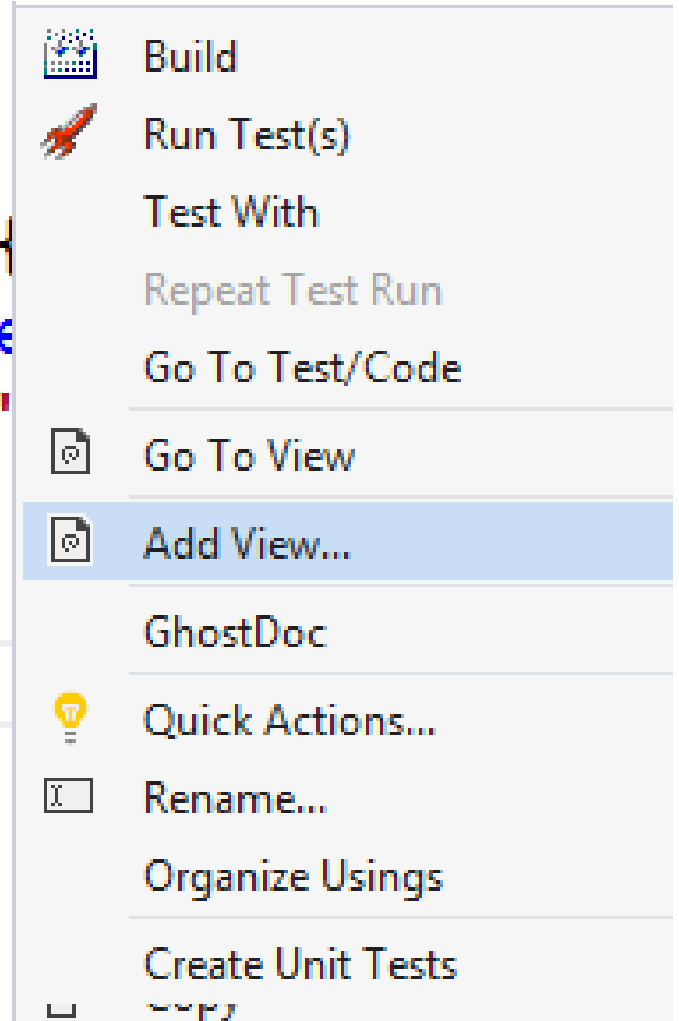
- Click droit
- Add View

```
Result Index()
```

```
InfoUser = new {  
InfoProduit = new {  
View("OtherIndex")
```

```
Result Edit()
```

```
View();
```



Les vues

Dans l'écran suivant , nous devons choisir les options de création :

- **View Name** : nom de la vue
- **Template** : Permet d'avoir une vue avec les actions usuels par rapport à un template (List, Edit, Create,...).

C'est ce qu'on appelle le *scaffolding*

The screenshot shows the 'Add View' dialog box. The 'View name' field contains 'Edit'. The 'Template' dropdown is open, showing 'Empty (without model)' as the selected option. The 'Model class' dropdown is also open, showing a list of options: 'Create', 'Delete', 'Details', 'Edit', 'Empty', 'Empty (without model)', and 'List'. The 'Data context class' dropdown is open, showing 'Empty'. The 'Options' section has three checkboxes: 'Create as a partial' (unchecked), 'Reference script lib' (unchecked), and 'Use a layout page:' (checked). Below the 'Use a layout page:' checkbox is a text box for the layout page name, with a button '...' to its right. Below the text box is the text '(Leave empty if it is set in a Razor _viewstart file)'. At the bottom right are 'Add' and 'Cancel' buttons.

Les vues

Scaffold	Description
Create	Une vue avec un formulaire permettant de créer une instance du Model. 1 label + 1 input pour chaque propriété
Delete	Une vue avec un formulaire permettant de supprimer une instance existante du Model. 1 label + la valeur courante de chaque propriété.
Details	Une vue affichant les valeurs de chaque propriété du Model.
Edit	Une vue avec un formulaire permettant de modifier une instance existante du Model. 1 label + la valeur courante de chaque propriété.
Empty	Vue vide. Seul le type du Model peut apparaitre.
Empty(without model)	Une vue vide. Seul template permettant de ne pas sélectionner de Model lors de la création.
List	Une vue affichant les propriétés du Type Model. Il faut lui passer un <i>Ienumerable<T></i> . Cette vue contient également un lien vers les actions <i>Create/Edit/Delete</i>

Les vues

- **Create as Partial View** : permet de créer une vue qui n'aura ni <head> ni <body> ni <html>. Elle est destinée à s'intégrer dans une autre.
- **Reference script libraries** : Permet d'intégrer les bibliothèques javascripts (jQuery,..) nécessaires à la vue.
Pour *Create et Edit*, cette option ajoute une référence au bundle *jqueryEval* qui est nécessaire pour la validation de formulaire côté client.
- **Use a layout page** : permet de spécifier le template visuel associé à la vue. Par défaut, il s'agit de la vue partielle : /Views/Shared/_layout.cshtml

The screenshot shows the 'Add View' dialog box. The 'View name' field contains 'Edit'. The 'Template' dropdown is set to 'Create'. The 'Model class' and 'Data context class' dropdowns are empty. Under the 'Options' section, the 'Use a layout page' checkbox is checked. Below this checkbox is a text box for specifying the layout page, followed by a browse button ('...'). A note indicates to leave it empty if set in a Razor _viewstart file. The 'Add' and 'Cancel' buttons are at the bottom right.