

ASP.NET

M. Person – Cognitic Sprl



ASP.NET : INTRO

Le but de la programmation ASP.NET est de réaliser des sites webs dynamiques caractérisés par une interaction poussée avec l'utilisateur ainsi que par l'accès à une base de données Partagée entre tous les visiteurs du site.

Programmation côté serveur ?

Les pages HTML que reçoit le client sont générées dynamiquement sur le serveur (à partir des balises ASP.NET) et le code C# est exécuté sur ce serveur. Le client ne reçoit que du Html et du Javascript.

Les avantages d'ASP.NET par rapport à ASP3

- le code n'est plus interprété mais compilé, gain de vitesse !
- le choix parmi plusieurs langages pour construire ses pages !
- le code ASP.net peut être séparé du code HTML servant de mise en page, clarté et efficacité !
- vous pouvez développer vos propres composants, les installer et les enregistrer sur le serveur par simple transfert FTP, sans le redémarrer, la prise en compte est immédiate !

ASP.NET : STRUCTURE GÉNÉRALE D'UNE PAGE ASP.NET

Une page asp.NET possède une extension .aspx

```
<head>
    <script language="c#" runat="server">
        public void fct (type I)
        {
            .....
        }
    </script>
    <script language="javascript">
        function f ;
    </script>
</head>
<body>
    <table> ... </table>
    <asp :Label [Propriétés]></asp :Label>
    <%=fct(5)%>
</body>
```

On peut séparer cette structure type en trois parties :

3

ASP.NET : STRUCTURE GÉNÉRALE D'UNE PAGE ASP.NET

Tout script écrit en C# pour asp.NET doit se trouver entre les balises suivantes

```
<script language="c#" runat="server"> ...  
</script>
```

Le tout devant se trouver entre les balises html ouvrante et fermante.

Nous verrons plus loin qu'il est possible de séparer ce code C# de la page asp (technique du *code-behind*)

Il est possible aussi d'appeler du code directement à l'intérieur de la page web.

```
<form runat="server">  
...  
<h2> Date du jour : <% =DateTime.Now.ToString() %> </h2>  
...  
</form>
```



Date du jour : 29/02/2004 22:57:53

ASP.NET : STRUCTURE GÉNÉRALE D'UNE PAGE ASP.NET

Il est également possible d'appeler des fonctions personnelles

```
<head>
    <script language="C#" runat="server">
        public string Demain()
        {
            DateTime Jour = DateTime.Now;
            Jour = Jour.AddDays(1)
            return Jour.ToString()
        }
    </script>
</head>
<body>
    <form runat="server">
        ...
        Date du jour : <% =DateTime.Now.ToString() %><br>
        Date de demain : <% =Demain() %> ...
    </form>
</body>
```



... Date du jour : 29/02/2004 23:07:01
Date de demain : 1/03/2004 23:07:01 ...

ASP.NET : LES COMPOSANTS SIMPLES

Label	Composant très simple permettant d'afficher du texte.
Button	Exécute des fonctions.
TextBox	Il s'agit d'une « case » que l'on peut compléter
Checkbox	A n'utiliser que lors d'un choix qui se limite à deux valeurs (Oui/Non, Homme/Femme...)
RadioButtonList	Permet de sélectionner un choix parmi une liste de choix possible.
ListBox	La place utilisée pour une telle liste est fixe
DropDownList	Il s'agit d'une liste déroulante qui se met au premier plan, par dessus les autres composants lorsqu'elle est déroulée

ASP.NET : LES COMPOSANTS SIMPLES

Image	Comme son nom l'indique, il s'agit d'une image.
ImageButton	Il s'agit d'un bouton qui a la forme d'une image.
Hyperlink	hyperlien
LinkButton	Il s'agit d'un bouton qui a la forme d'un hyperlien.
Panel	Ce composant est intéressant pour afficher ou non un nombre important de composants d'un coup.
Calendar	Il s'agit d'un calendrier qui a la forme qui est la plus utilisée.

Propriétés communes

- ***Id*** → Tout composant peut porter un nom, ce nom est l'Id. C'est à partir de ce nom que nous pourrons modifier ses attributs dans du code VB.NET .
On ne peut donc avoir deux fois le même Id dans une même page asp.NET.
- ***OnClick*** → est l'événement lorsque l'on clique sur un composant, par exemple un bouton
- COGNITIC**
- ***Runat*** → Cet attribut permet de spécifier où le traitement du composant doit être fait.

ASP.NET : LES COMPOSANTS SIMPLES

Text → est la valeur affichée par le composant.

Cette valeur peut inclure des balises html qui seront considérées en tant que telles.

-**OnTextChanged** → est l'événement lorsque le texte change. Comme pour le **OnClick**, il faut donc spécifier une méthode de traitement et lui affecter son nom

-**AutoPostBack** → Si sa valeur est à **true**, la page est automatiquement renvoyée au serveur pour qu'il puisse faire son traitement. Dans le cas contraire, la page ne sera envoyée que lors d'un clic sur un bouton.

ASP.NET : LES LABELS

Ce premier composant est une zone d'affichage.

Sa syntaxe est la suivante :

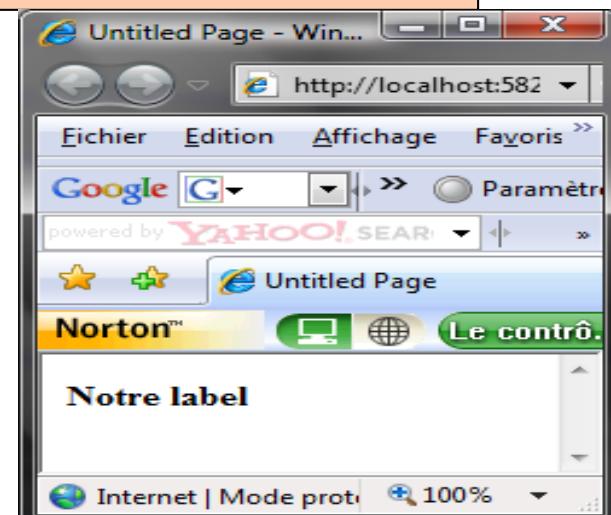
```
<asp:Label id="nom_compo" text="text_label"  
runat="server"/>
```

Le composant sera compilé par le serveur et traduit en html avant d'être envoyé au client.

Il est possible de passer des balises html dans la valeur du texte

```
<asp:Label id="zaValeur" text="Notre label" runat="server"/>
```

Ce composant n'a pour but que d'afficher du texte sur la page ...



ASP.NET : LES BUTTONS

Voici un exemple de syntaxe d'une balise

Button :

```
<asp:Button id="btnEnvoi" text="Go" OnClick="Envoi()"  
runat="server"/>
```

Nous pouvons remarquer que le bouton appelle la fonction Envoi.

Il faudra donc l'écrire

```
<head>  
<script language="C#" runat="server">  
    public void Envoi(sender as Object ,E as EventArgs)  
    {  
        nom_compo.Text="Notre deuxième composant asp  
fonctionne"  
    }  
</script>  
</head>  
<body>  
<form runat="server">  
    <asp:Label id="nom_compo" text="Notre composant  
asp" runat="server"/>  
    <asp:Button id="btnEnvoi" text="Go" OnClick="Envoi" runat="server"/>  
</form>
```

ASP.NET : LES BUTTONS



ASP.NET : LE TEXTBOX

Une zone d'édition est une zone de texte qui peut être modifiée par l'utilisateur

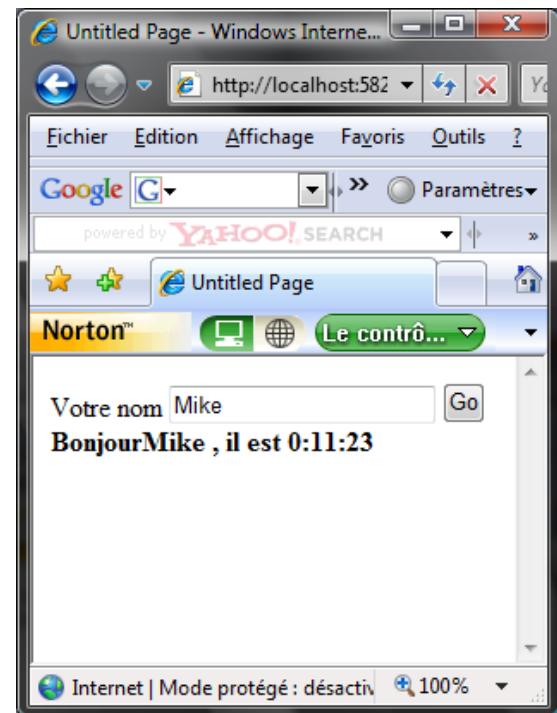
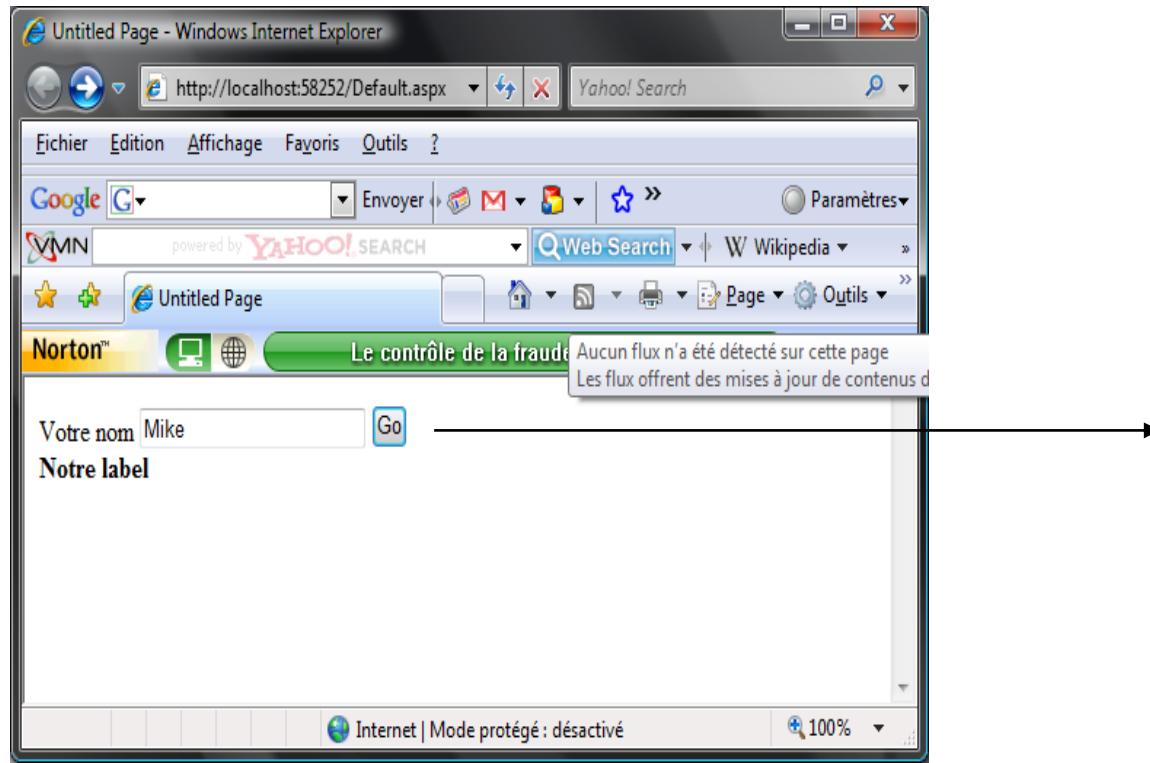
```
<asp :TextBox id="txtnom" runat="server"/>
```

Propriétés

- **Columns** : il s'agit de la largeur du contrôle exprimée en nombre de caractères.
- **MaxLength** : est le nombre maximal de caractères qui seront admis.
- **Rows** : est le nombre de lignes visibles. On peut saisir plus de lignes mais dans ce cas, il y aura défilement vertical
- **TextMode** : Les trois modes qui sont possibles sont **Single**, **MultiLine** ou **Password**.
- **Wrap** : permet de stipuler si oui ou non il y passage automatique à la ligne dans une zone d'édition de plusieurs lignes

ASP.NET : PREMIER EXERCICE

- Mettre en place le formulaire suivant



ASP.NET : PREMIER EXERCICE

- **Deuxième exercice**

Mettre en place un formulaire de contact

The image shows a screenshot of an ASP.NET contact form. It consists of six input fields arranged vertically, each preceded by a label and an asterisk indicating it is required. At the bottom is a large blue "Envoi!" button.

Utilisateur : (*)	<input type="text"/>
Nom :	<input type="text"/>
Prenom : (*)	<input type="text"/>
Email : (*)	<input type="text"/>
Mot de passe : (*)	<input type="text"/>
Confirmation Mot de passe : (*)	<input type="text"/>
Envoi !	

Vérification :

- nom d'utilisateur de 5 à 10 caractères
- nom de famille facultatif avec une longueur < 25 caractères
- Vérifier l'email
- Mot de passe de min 3 lettres + 2 chiffres + 1 @
- mot de passe = confirmation mot de passe

ASP.NET : CHECKBOX (CASE À COCHER)

Une case à cocher permet de faire un choix de type oui/non ou vrai/faux. Comme toujours, ce contrôle possède au minimum un id. Il a également comme propriétés :

- *Checked* : est à *true* si la case est cochée et à *false* dans le cas contraire
- *TextAlign* : est la position du libellé par rapport à la case. Les valeurs possibles sont *Right* ou *Left*

```
<asp:CheckBox id="ccMailingList" runat=server" text="S'inscrire à la mailing-list"/>
```

ASP.NET : RadioButtonList (CASES À COCHER)

Parmi plusieurs cases à cocher d'un même ensemble, on ne peut en choisir qu'une seule, ce qui implique que lorsque l'on cocher une autre case, celle qui était au préalable cochée est décochée. On les utilise donc lorsque l'on a plusieurs choix mais qu'une seule valeur ne peut être choisie.

Pour les propriétés, elles sont identiques à un CheckBox si ce n'est que ces cases à cocher doivent être dans un même groupe. On lui donne donc un nom (propriété GroupName).

```
<asp:RadioButton id="Madame" runat="server" value="Madame"  
GroupName="rblMmeMlleMr"/>  
<asp:RadioButton id="Mademoiselle" runat="server" value="Mademoiselle"  
GroupName="rblMmeMlleMr"/>  
<asp:RadioButton id="Monsieur" runat="server" value="Monsieur"  
GroupName="rblMmeMlleMr"/>
```

ASP.NET : LISTBOX (BOÎTE DE LISTE)

Une boîte de liste est un composant permettant d'afficher une liste de valeur possible.

Les caractéristiques de ce composant sont les suivantes :

- *SelectionMode* qui indique l'on peut sélectionner un (*Single*) ou plusieurs articles (*Multiple*) simultanément
- *Items* : qui est la collection des articles
- *SelectedIndex* : qui est le numéro de l'article sélectionné, le premier ayant comme valeur 0
- *SelectedItem* : est l'article sélectionné
- *SelectedItems* est la collection des articles sélectionnés
- *Rows* est le nombre de lignes pour la liste visibles

```
<asp:ListBox id="lbPays" runat="server" SelectionMode="Single" Rows="2">
    <asp:ListItem text="Belgique" runat="server"/>
    <asp:ListItem text="France" runat="server"/>
    <asp:ListItem text="Suisse" runat="server"/>
</asp:ListBox>
```

ASP.NET : DROPODOWNLIST (BOÎTE COMBO)

La boîte com
bo possède les mêmes propriétés qu'une ListBox.

```
<asp:DropDownList id="ddlMoisNaissance" runat="server" SelectionMode="Single"  
Rows="10">  
    <asp:ListItem text="Janvier" runat="server"/>  
    <asp:ListItem text="Fevrier" runat="server"/>  
    ...  
    <asp:ListItem text="Octobre" runat="server"/>  
    <asp:ListItem text="Novembre" runat="server"/>  
    <asp:ListItem text="Decembre" runat="server"/>  
</asp:DropDownList>
```

ASP.NET: IMAGE

On retrouve les propriétés suivantes :

- *ImageUrl* qui est l'adresse de l'image (relative ou absolue)
- *AlternateText* est le texte à afficher si l'image ne peut l'être
- *ImageAlign* est l'alignement de l'image par rapport au contour de son composant.
On retrouve comme valeurs acceptées AbsBottom, AbsMiddle, BaseLine, Bottom, Left, Middle, NotSet, Right, TextTop et Top.

```
<asp:Image ImageUrl="dotnet.gif" runat="server" AlternateText="image"/>
```

ASP.NET : IMAGEBUTTON

D'apparence, ce composant est le même que le précédent, si ce n'est qu'un événement lui est associé lors du clic.

Pour ce qui est des propriétés, on ne retrouve que *ImageUrl* par rapport au précédent.

```
<asp:ImageButton ImageUrl="dotnet2.gif" runat="server" AlternateText="Une info"  
OnClick="ibClick"/>
```

Et donc nous retrouverons un code c# gérant ce click

```
private void ibClick(object sender, ImageClickEventArgs e)  
{  
...  
}
```

ASP.NET : HYPERLINK (HYPER-LIEN)

Les propriétés sont simples :

- *ImageUrl* est l'image qui contient le lien (si l'on désire utiliser une image comme hyperlien). Si cette propriété est utilisée, il ne faut pas utiliser la propriété *Text*.
- *NavigateUrl* est l'adresse destination

```
<asp:Hyperlink Text="Site dynamique" NavigateUrl="http:\\www.dynatic.be"  
runat="server"/>
```

ASP.NET : CALENDAR (CALENDRIER)

Il s'agit d'un calendrier complet prêt à l'emploi.

On trouve une liste interminable (mais logique) de propriétés pour ce composant.

Si il y en a deux à retenir, il s'agit bien de celles-ci :

- *SelectedDate* qui donne la date sélectionnée
- *VisibleDate* qui est la date visible

Ces deux éléments sont de type *DateTime*.

```
<asp:Calendar VisibleDate="2003/12/11" runat="server"/>
```

La date est de type anglosaxone.

décembre 2003						
lun.	mar.	mer.	jeu.	ven.	sam.	dim.
<u>24</u>	<u>25</u>	<u>26</u>	<u>27</u>	<u>28</u>	<u>29</u>	<u>30</u>
<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>
<u>8</u>	<u>9</u>	<u>10</u>	<u>11</u>	<u>12</u>	<u>13</u>	<u>14</u>
<u>15</u>	<u>16</u>	<u>17</u>	<u>18</u>	<u>19</u>	<u>20</u>	<u>21</u>
<u>22</u>	<u>23</u>	<u>24</u>	<u>25</u>	<u>26</u>	<u>27</u>	<u>28</u>
<u>29</u>	<u>30</u>	<u>31</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>

ASP.NET : EXERCICE

- **Mettre en place un formulaire d'inscription à partir du formulaire créé lors du deuxième exercice.**
 - Choix dans de la ville dans une listbox
 - choix du pays dans une combobox
 - Encodage de la date de naissance grâce à un calendrier
 - lien vers une charte d'utilisation (charte.html)
 - Vérification de l'e-mail
 - Choix d'inscription à la news-letter
 - Choix via trois possibilité pour le titre → Mr.,Mme,Mlle

ASP.NET : LES ÉVÉNEMENTS PARTICULIERS

1. Application

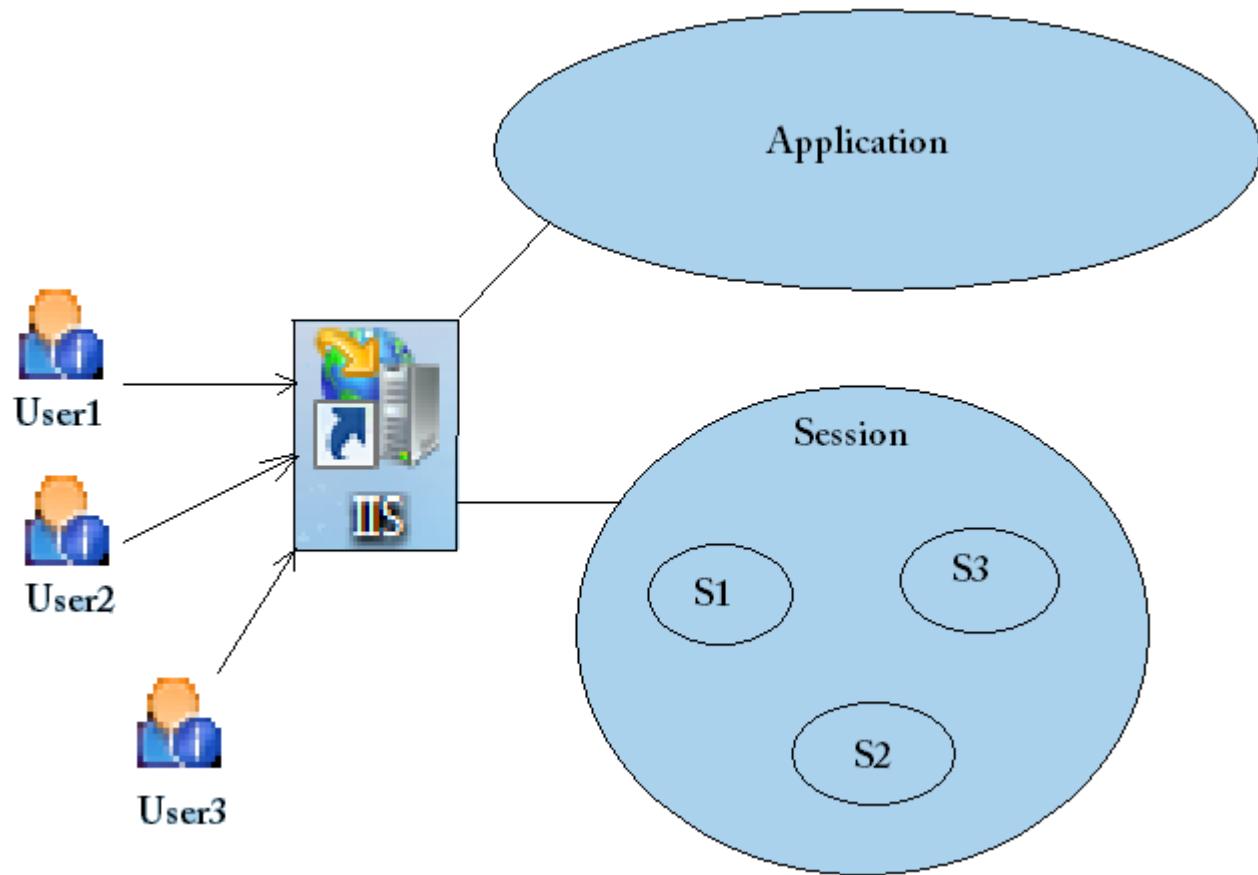
<u>Evénement</u>	<u>Description</u>
Application_Start	Exécuté après le premier appel à une page du site depuis le démarrage de IIS
Application_End	Appelé lorsque l'application se termine, cela ne signifie pas que IIS s'arrête mais est d'office appelé si, pour une raison quelconque IIS est arrêté

2. Session

<u>Evénement</u>	<u>Description</u>
Session_Start	Appelé lors de chaque nouvelle session d'un navigateur client
Session_End	Fin de session soit explicite (Session.Abandon()) ou suite à un time out

ASP.NET : LES SESSIONS

- Différence entre Session et Application



ASP.NET : LES SESSIONS

Soit trois utilisateurs U1, U2 et U3 qui envoient une requête vers le serveur IIS.

Il y aura **un seul objet "Application" commun** à tous les utilisateurs du site mais **trois objets "Session"** correspondant chacun à un utilisateur précis.

Si U2 quitte son poste de travail sans couper son navigateur :

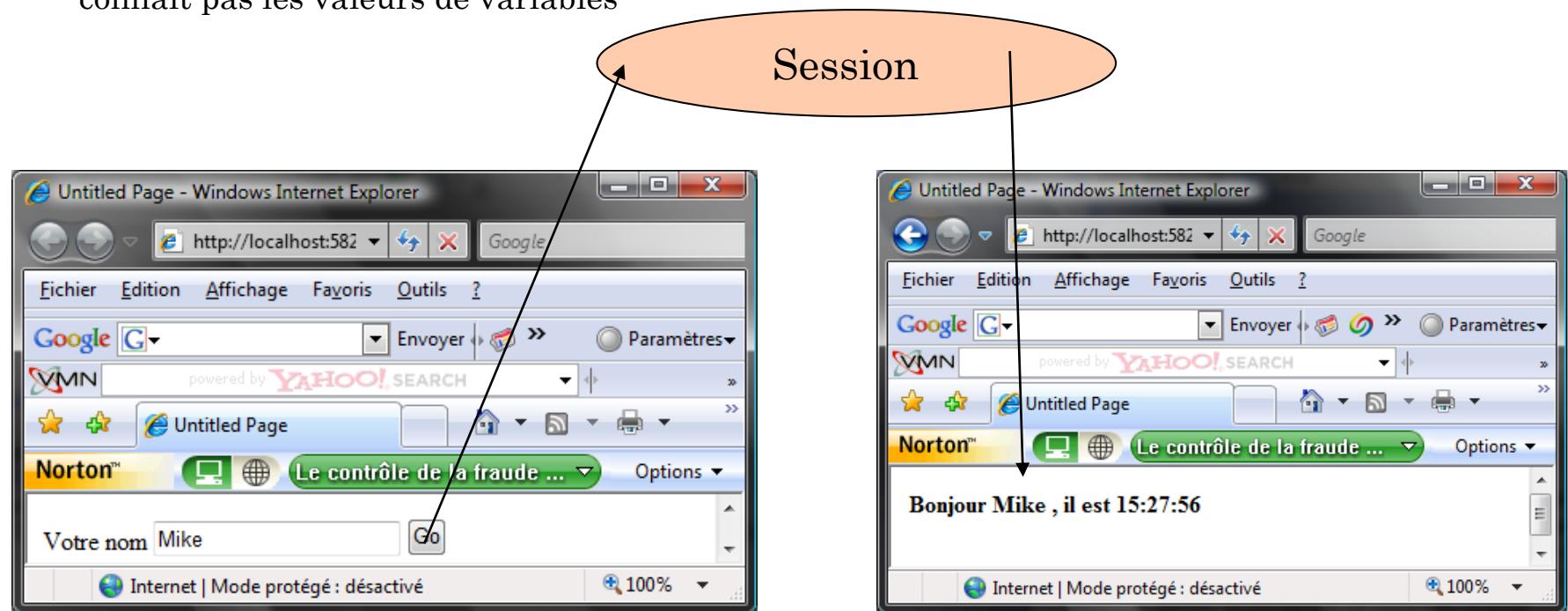
- S'il n'y a pas de timeout, les autres utilisateurs peuvent accéder à S2
- S'il y a timeout et que U2 revient visiter le site, une nouvelle session S4 sera créée

Par contre, si U2 coupe son navigateur, S2, persiste jusqu'à un éventuel timeout ou jusqu'à la fin de l'application

ASP.NET : LES SESSIONS

- Pourquoi des Sessions ?

Les sessions ASP.NET vous permet de stocker et de récupérer des valeurs pour un utilisateur à mesure que ce dernier navigue dans les différentes pages ASP.NET qui composent une application Web. HTTP est un protocole sans état, ce qui signifie que votre serveur Web traite chaque demande de page HTTP comme une demande indépendante ; par défaut, le serveur ne connaît pas les valeurs de variables



ASP.NET : LES SESSIONS

- **Comment utiliser les session ?**

La session ASP.NET est activé par défaut pour toutes les applications ASP.NET. Les variables d'état de session ASP.NET sont définies et récupérées facilement à l'aide de la propriété Session qui stocke les valeurs de variables de session comme une collection indexée par nom.

Par exemple, l'exemple de code suivant crée les variables de session **FirstName** et **LastName** pour représenter le prénom et le nom d'un utilisateur et les définit sur les valeurs récupérées des contrôles TextBox.

```
Session["FirstName"] = FirstNameTextBox.Text
```

```
Session["LastName"] = LastNameTextBox.Text
```

ASP.NET stocke par défaut les informations sur la session dans l'espace mémoire de l'application ASP.NET. Vous pouvez stocker éventuellement les informations sur la session à l'aide d'un service autonome afin qu'elles soient conservées, en cas de redémarrage de l'application ASP.NET, dans un serveur SQL Server

ASP.NET : LES SESSIONS

- **Comment changer l'endroit de stockage des sessions ?**

Vous pouvez spécifier le mode que l'état de session ASP.NET doit utiliser en assignant une valeur d'énumération **SessionStateMode** à l'attribut **mode** de l'élément **sessionState** dans le fichier **Web.config** de votre application.

→ **Mode InProc**

Il s'agit du mode par défaut : les variables de sessions sont stockées en mémoire sur le serveur web

→ **Mode Sql server**

Le mode **SQLServer** stocke l'état de session dans une base de données SQL Server.

L'utilisation de ce mode garantit que l'état de session est conservé en cas de redémarrage de l'application Web et qu'il est disponible pour plusieurs serveurs Web dans une batterie de serveurs Web.

```
<configuration>
    <system.web>
        <sessionState mode="SQLServer"
sqlConnectionString="Integrated Security=SSPI;data source=SampleSqlServer;" />
    </system.web>
</configuration>
```

ASP.NET : LES SESSIONS

Pour utiliser le mode SQLServer, vous devez d'abord être sûr que la base de données d'état de session ASP.NET est installée sur SQL Server.

Remarque :

Vous pouvez installer la base de données d'état de session ASP.NET à l'aide de l'outil Aspnet_regsql.exe

Pour configurer une application ASP.NET afin qu'elle utilise le mode SQLServer, procédez comme suit dans le fichier Web.config de l'application :

- Définissez l'attribut mode de l'élément **sessionState** avec la valeur SQLServer.
- Définissez l'attribut sqlConnectionString avec une chaîne de connexion pour votre base de données SQL Server.

ASP.NET : LES SESSIONS

- Installation de la base de données d'état de session à l'aide de l'outil Aspnet_regsql.exe

Pour installer la base de données d'état de session sur SQL Server, exécutez l'outil Aspnet_regsql.exe présent dans le dossier

systemroot\Microsoft.NET\Framework\numéroversion de votre serveur Web.

Fournissez les informations suivantes avec la commande :

- Nom de l'instance du serveur SQL Server à l'aide de l'option -S.
- Informations d'identification de connexion pour un compte ayant l'autorisation de créer une base de données sur SQL Server. Utilisez l'option -E pour utiliser l'utilisateur actuellement connecté, ou l'option -U pour spécifier un ID utilisateur avec l'option -P permettant de spécifier un mot de passe.
- Option de ligne de commande -ssadd pour ajouter la base de données d'état de session.

Par défaut, vous ne pouvez pas utiliser l'outil Aspnet_regsql.exe pour installer la base de données d'état de session sur SQL Server Express. Pour exécuter l'outil Aspnet_regsql.exe et installer une base de données SQL Server Express, vous devez d'abord activer l'option SQL Server Agent XPs à l'aide des commandes T-SQL, comme illustré ci-après :

ASP.NET: LES SESSIONS – BONNE PRATIQUE

- **Bonne pratique pour l'utilisation des sessions**

Pour placer un objet dans une Session d'un utilisateur, nous procédons ainsi :

```
Session["NomUtilisateur"] = "Richard"
```

- Il y a plusieurs inconvénients :
 - Si la variable NomUtilisateur est utilisé dans plusieurs pages, il faut bien savoir qu'elle s'appelle NomUtilisateur, il ne faut pas faire de fautes de frappes et surtout être sur que cette variable n'existe pas déjà dans une autre page avec une utilisation complètement différente. De plus, si vous voulez changer le nom de la variable en LeNomDelUtilisateur, vous devez revoir tout votre code de toutes vos pages,
 - La variable est non typé (c'est un Object dans Session) donc quand vous voulez y accéder, vous êtes obligés d'effectuer des opérations de casting :

```
String sM = (String)Session["NomUtilisateur"]
```

Qui dit casting dit possibilité d'erreur de casting.

ASP.NET: LES SESSIONS – BONNE PRATIQUE

Donc, l'idée est de créer une classe tampon qui accédera aux variables sessions

```
public class Utils
{
    public static string NomUtilisateur
    {
        get{return HttpContext.Current.Session["NomUtilisateur"];}
        set{HttpContext.Current.Session["NomUtilisateur"] = value;}
    }
}
```

Dans nos pages nous pouvons désormais écrire :

```
String sM = Utils.NomUtilisateur
```

Pas de casting, un seul endroit pour gérer le nom des variables sessions, etc.

ASP.NET : GET

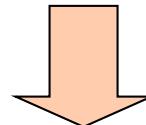
- QueryString (Get)

ou "chaîne de requête". Permet de faire passer des informations via l'URI d'une page à une autre.

Prenons l'exemple d'un ID de société sélectionné dans une page dont les données sont présentées dans une autre page, on aurait très bien pu indiquer cet ID via l'URI lors de l'appel à la deuxième page.

Vous avez choisi la société ayant un ID = 1235, voici comment passer l'identifiant à la page suivante :

```
<A href="page2.aspx?idsoc=1235"></A>
```



```
<p>Vous avez choisi la société : +  
Request.QueryString["idsoc"]</p>
```

ASP.NET : POST

Excepté pour le téléchargement de fichier, l'objet Post n'est plus utilisé en ASP.NET. En effet, désormais, ASP.NET étant orienté à 100% objet, tout formulaire utilise les événements.

Ceux-ci étant exécuté côté serveur, il n'est donc plus utile de choisir la méthode post puisque les données sont transmises en arrière plan au serveur.

Nous verrons cependant par la suite, comment utiliser l'objet ***HttpPostedFile*** afin d'uploader un fichier via un formulaire

ASP.NET : EXERCICE

- A partir du formulaire mis en place dans les exercices précédents, mettre en place l'inscription de l'utilisateur dans la base de donnée Web sur sql server.
- Lorsque la personne est inscrite , redirigez la vers la page membre.aspx qui ne peut être accédée que si l'utilisateur est authentifié (pour cela utiliser les sessions)
- Mettre en place un formulaire « connectez-vous »

ASP.NET: LES CONTRÔLES DE VALIDATION

Lorsque nous désirons contrôler un formulaire, il est souvent fastidieux et long de mettre en place le code C# ou Javascript contrôlant chacun des contrôles.

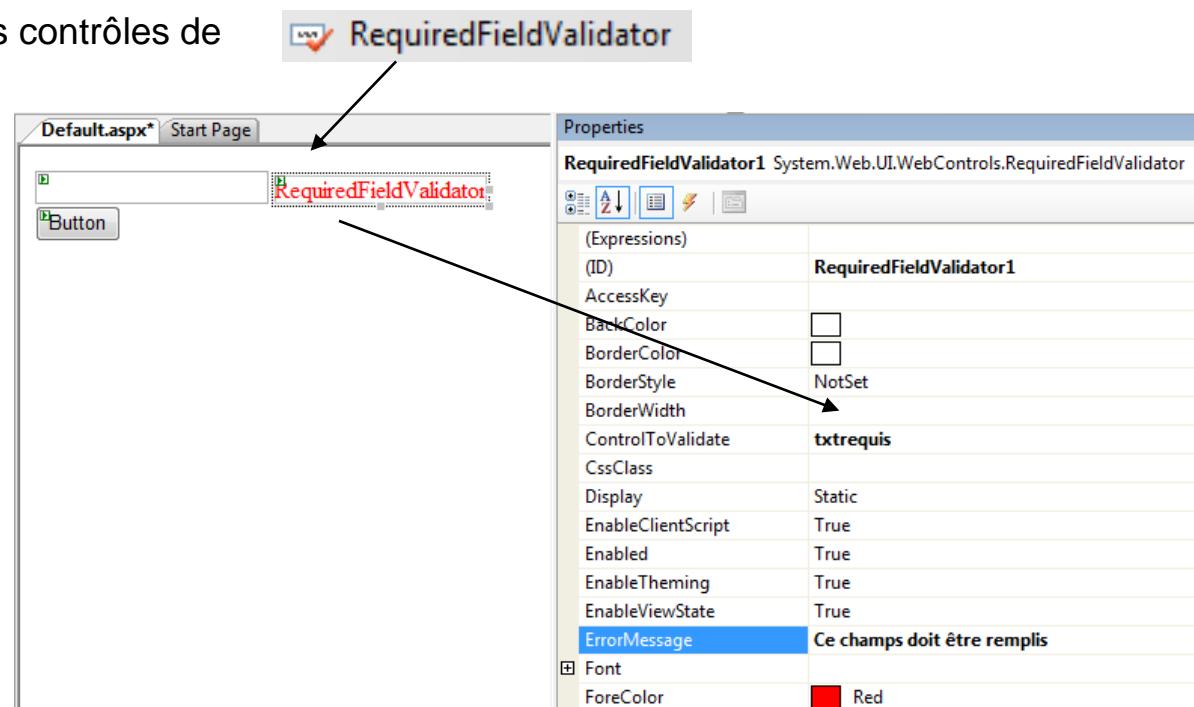
.Net vient à notre secours avec les contrôles de validation

- **Required Field Validator**

Ce contrôle est utilisé pour les champs qui doivent être remplis.

Si le contrôle n'est pas remplis, le *Required field Validator* stoppe le process et

attend que celui-ci soit rempli.

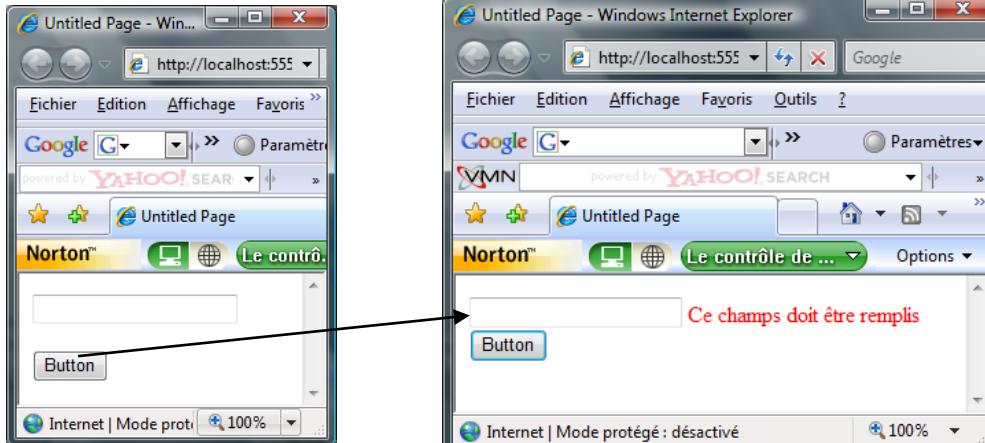


ASP.NET: LES CONTRÔLES DE VALIDATION

Il nous reste encore à placer le code dans notre événement du bouton

```
protected Button1_Click (object sender, EventArgs e)
{
    if page.IsValid
    {
        'code de gestion du formulaire
    }
}
```

Page.IsValid nous renvoie true si aucun des contrôles de validation n'est déclenché, false dans le cas inverse



ASP.NET: LES CONTRÔLES DE VALIDATION

• Compare Validator

Nous permet de comparer la valeur de deux champs

codersource

codersouce

Button

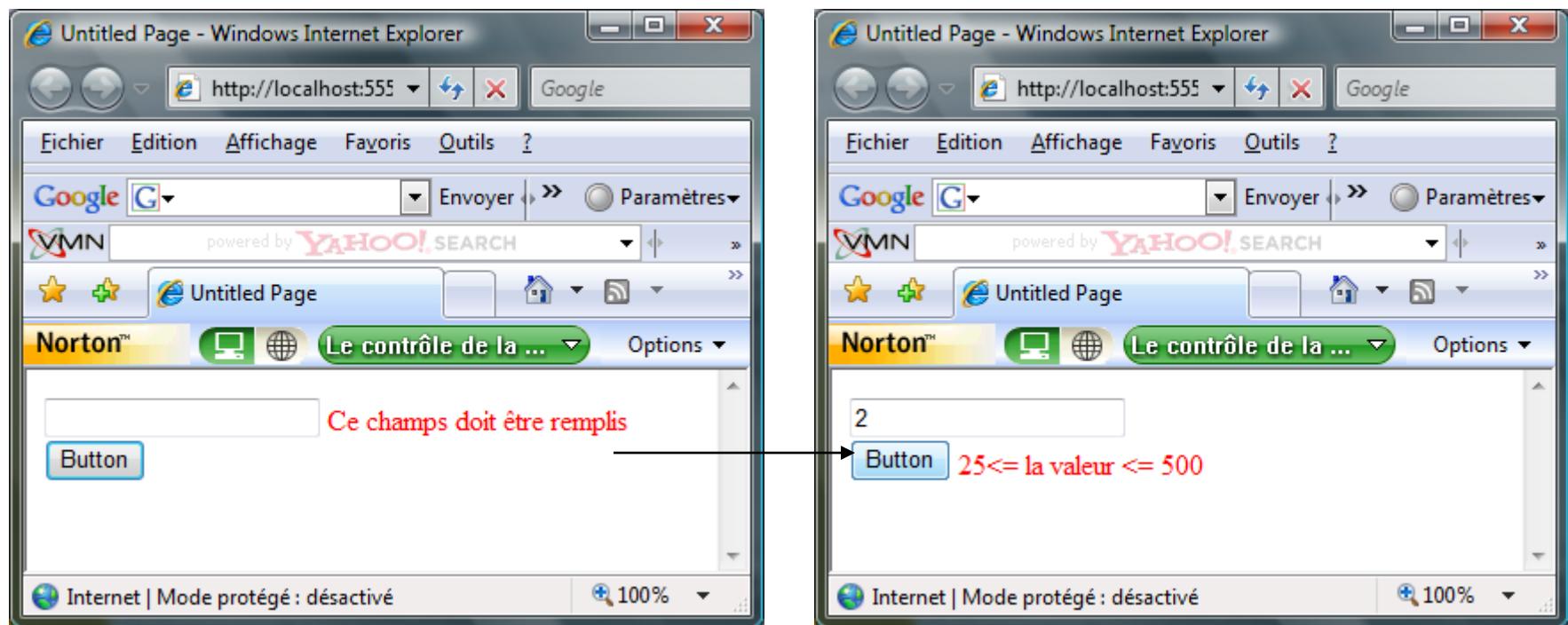
Not Equal

• Range Validator

Ce contrôle est utilisé lorsque l'on désire vérifier que la valeur encodée est comprise entre deux bornes (MinimumValue et MaximumValue)

```
protected void Page_Load(Object sender , System.EventArgs e)
{
    RangeValidator1.MinimumValue = 25
    RangeValidator1.MaximumValue = 500
    RangeValidator1.ControlToValidate =txtrequis.ID
    RangeValidator1.ErrorMessage = "25<= la valeur <= 500"
}
```

ASP.NET: LES CONTRÔLES DE VALIDATION

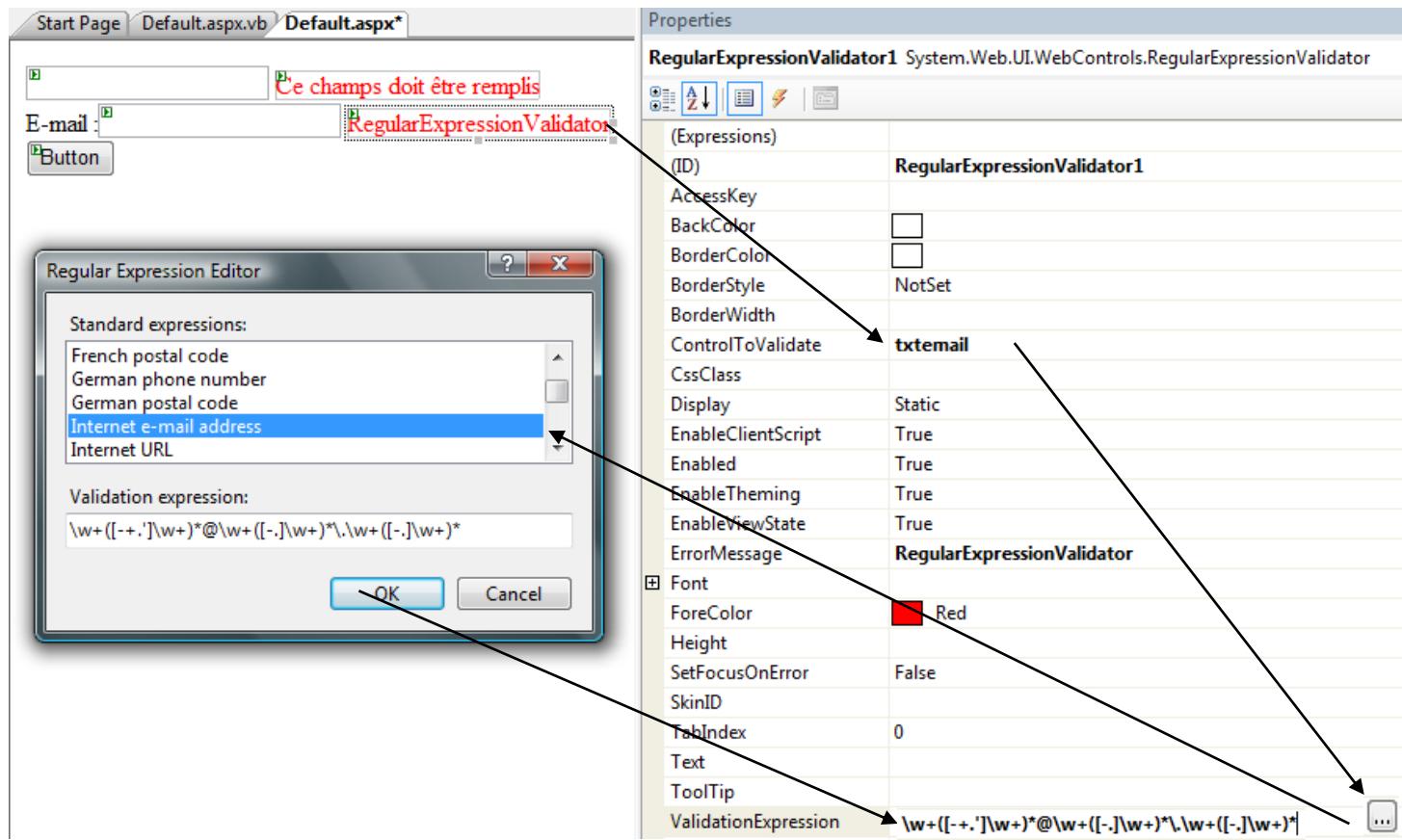


ASP.NET: LES CONTRÔLES DE VALIDATION

• Regular expression Validator

Nous permet de vérifier un champs grâce aux expression régulières.

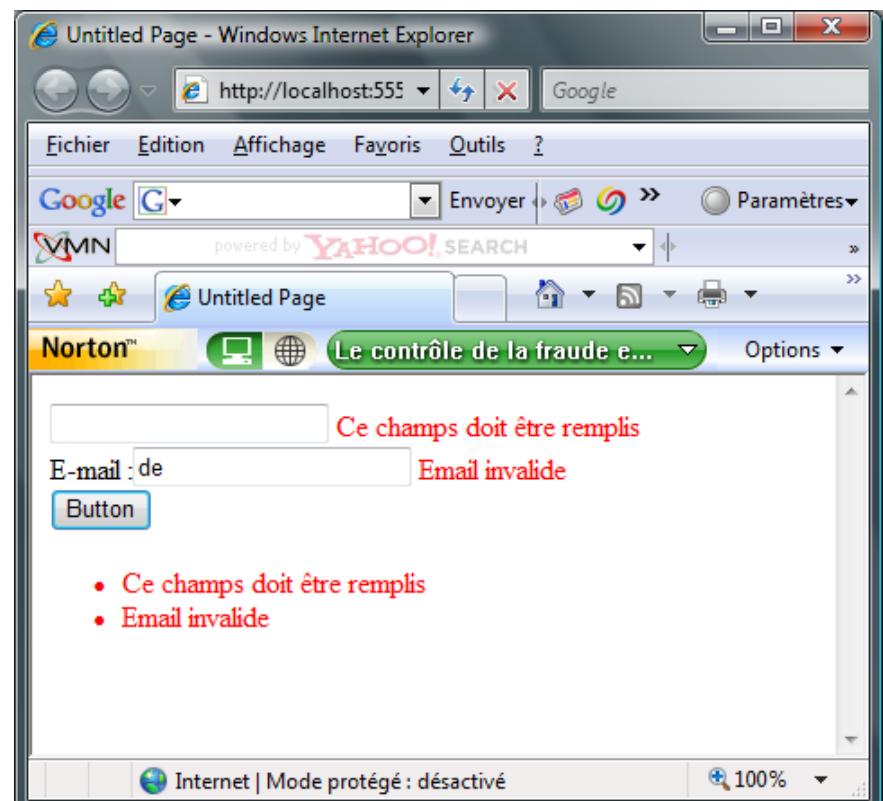
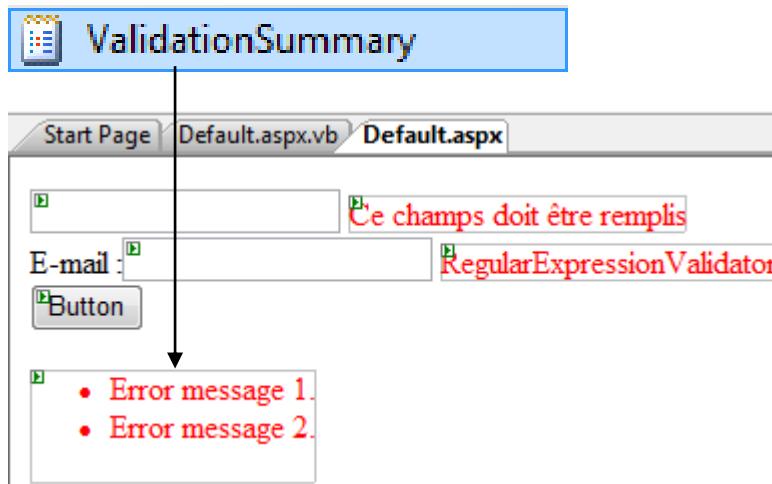
Par exemple pour contrôler un mail, numéro de téléphone Us, Code postal Us, carte de crédit,...etc..



ASP.NET: LES CONTRÔLES DE VALIDATION

• Validation summary Contrôle

Il est possible via ce contrôle de rassembler tout les messages d'erreur en un seul endroit. En bref, avoir un résumé des erreurs.



ASP.NET : EXERCICE

- Reprenez l'exercice précédent et mettez en place les validations via les validator
- Mettre en place une validation d'un formulaire.
Le formulaire possède les champs suivants
Les deux champs pseudos doivent être semblables
L'année de naissance doit être entre 1900 et 1988
L'e-mail doit être vérifié grâce à l'expression régulière suivante :
`\w+([-.] \w+)*@\w+([-.] \w+)*\.\w+([-.] \w+)*`

Le customValidator vérifie grâce à une fonction si le nombre entrée est paire.

COGNITIC

Exemples de validation

The screenshot shows a series of input fields and their validation status:

- Nom : Admin (Valid, green)
- Prénom : (Valid, green)
- Pseudo (Valid, green)
- Retaper le pseudo (Valid, green)
- Année de naissance (Valid, green)
- Adresse mail (Valid, green)
- Exemple de CustomValidator :* (Valid, green)
- Entrez un nombre pair : (Invalid, red)

A tooltip for the invalid field contains the error messages:

- Error message 1.
- Error message 2.

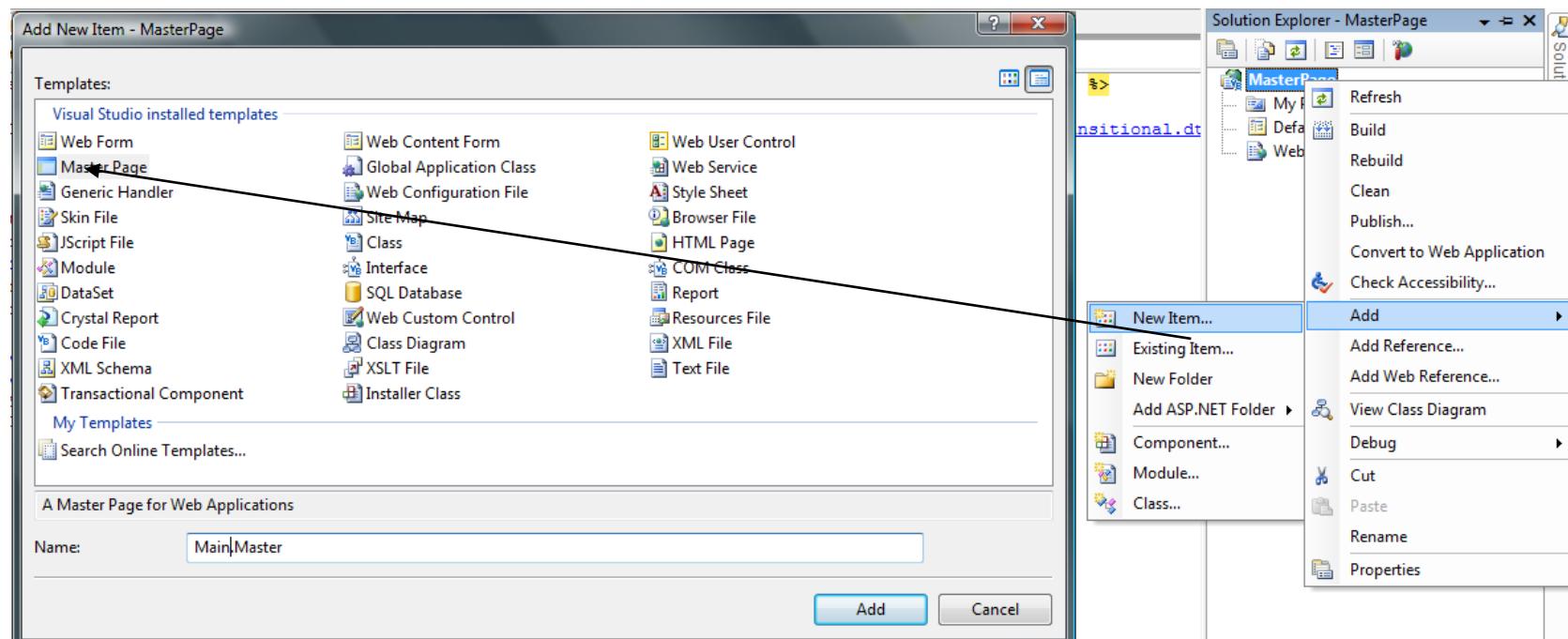
At the bottom, there is a "Valider" button and a status message "En cours de validation !".

ASP.NET : LES MASTER PAGES

Les **Master Page** nous permettent de créer un modèle (*template*) générique pour notre site web qui sera appliqué aux pages auxquelles vous avez décidé de l'appliquer.

L'avantage de ce système est de ne pas devoir ajouter à chaque page les éléments répétitifs qui les composent.

Nous pouvons comparer ces **master page** au principe des *includes* en php



ASP.NET : LES MASTER PAGES

Nous pouvons maintenant y ajouter les éléments dont nous avons besoin pour créer notre modèle.
Pour cela, il est préférable de passer en mode Design.

Pour placer et définir les éléments que vous désirez, vous allez utiliser les "*ContentPlaceHolder*".



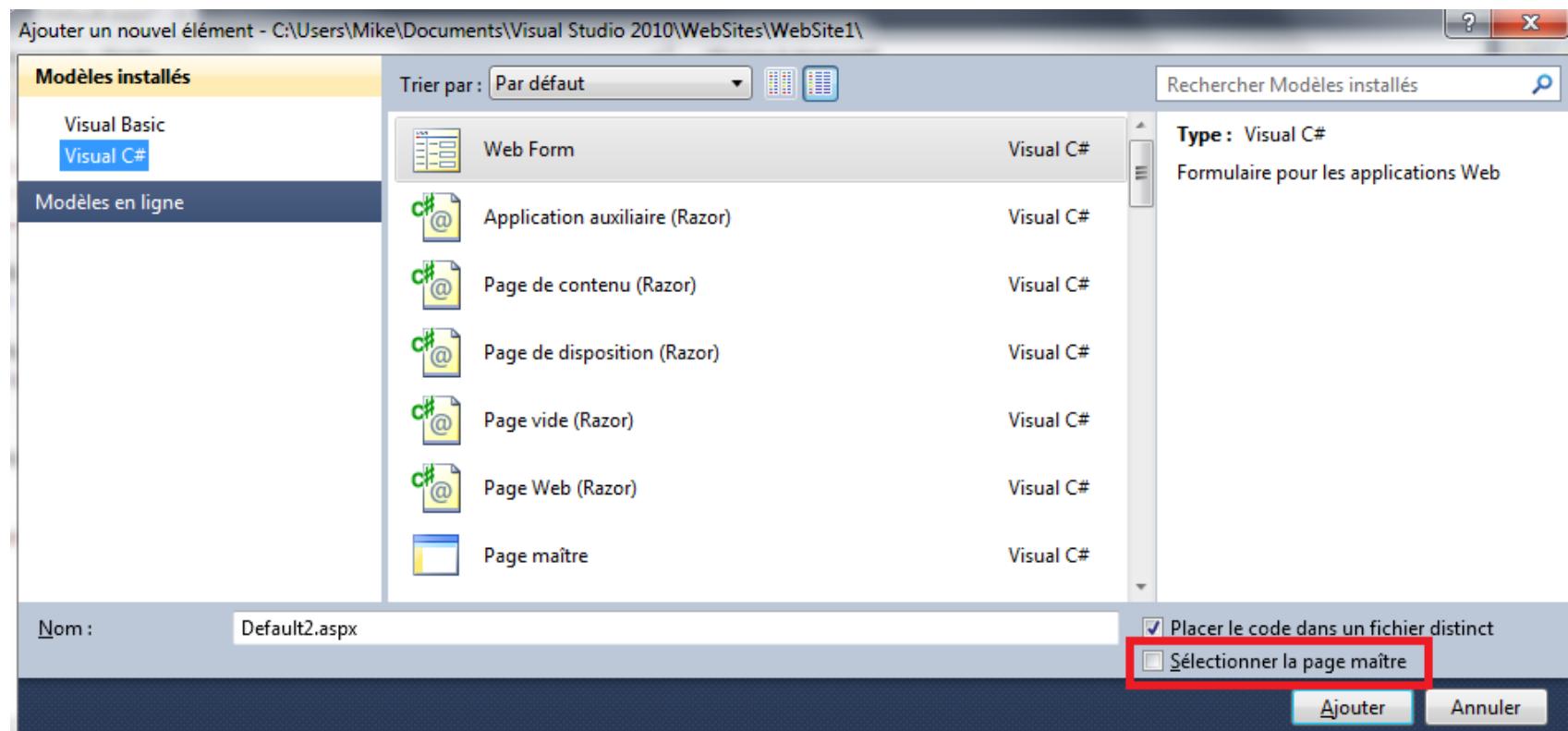
Une fois notre master créée, il nous suffit maintenant de spécifier celle-ci pour les pages de notre site.
Soit directement dans le code pour une page déjà créée :

```
<%@ Page Title="Page d'accueil" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
    CodeFile="Default.aspx.cs" Inherits="_Default" %>
<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="ContentPlaceHolder1">
```

A dropdown menu is open next to the "MasterPageFile" attribute, showing the option "Site.master" highlighted in yellow, indicating it is the selected master page for the current page.

ASP.NET : LES MASTER PAGES

Soit lors de la création de la page :



ASP.NET : LES MASTER PAGES

- **Exercice**

Mettre en place une master page du design suivant :



Dynatic (c)Copyright 2007

Une fois votre master page développée, appliquez là à tout les exercices précédents.

Pour ce faire, faites en sorte de n'avoir qu'un projet asp contenant le formulaire d'inscription, une page d'accueil avec le formulaire de login et bien entendu la zone Membre qui contiendra le formulaire de l'exercice de validation

ASP.NET & DESIGN

Depuis ASP.NET 2.0, il est possible de développer vos sites et sur le côté d'écrire une série d'apparences, appelés thèmes.

Ces thèmes permettent de définir l'apparence des pages et des contrôles afin de les appliquer facilement à une ou plusieurs pages voir à toute votre application web.

Un thème se compose d'un ensemble d'éléments qui sont :

- Les apparences
- Les feuilles de style (CSS)
- Les images et autres ressources comme des fichiers textes

ASP.NET & DESIGN

Les apparences

Les apparences permettent de paramétriser un ensemble de propriétés pour un ou plusieurs contrôles.

Cependant, seules les propriétés ayant l'attribut **ThemeableAttribute** à **true** peuvent être redéfinies.

Il existe deux types d'apparences de contrôle :

- **L'apparence par défaut :**

On définit un ensemble de propriétés pour un contrôle et si on applique le thème à une page, l'apparence va s'appliquer à tous les contrôles du même type de la page en question

- **L'apparence nommée :**

L'apparence nommée va être identifiée par un nom qui sera placé dans la propriété **SkinID**. Tous les contrôles du même type d'une page web qui auront comme **SkinID** le même nom auront donc l'apparence définie.

Pour définir un ensemble d'apparences pour un thème, on va utiliser un fichier de type **Skin** (extension **.skin**).

ASP.NET & DESIGN

Les feuilles de style CSS

En plus de définir des apparences dans un thème, vous pouvez y ajouter vos propres feuilles de style (CSS)

Les images et ressources

Il est aussi possible d'inclure des fichiers ressources liés à un thème, comme des images, des fichiers textes, audio, vidéo, etc.

En général, les fichiers ressources se trouveront dans un sous dossier de votre dossier des thèmes mais ce n'est pas une obligation.

ASP.NET & DESIGN

La portée des thèmes

Il existe deux types de thèmes ayant des portées différentes.

-Thèmes de page

il s'agit d'un répertoire thème constitué de fichiers d'apparences de contrôles, de feuilles de styles et de ressources

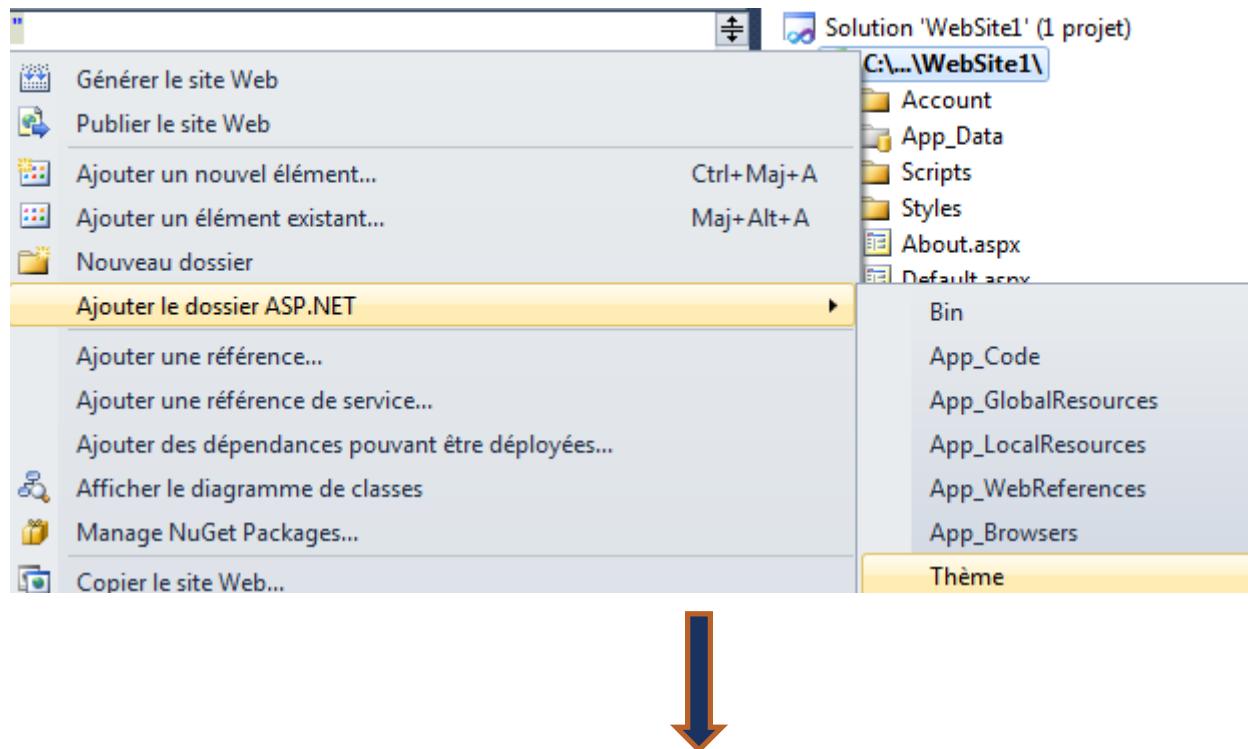
-Thèmes globaux

Il s'agit de thèmes qui peuvent être appliqués à tous les sites webs du serveur.
A la différence des thèmes de pages, les thèmes globaux sont stockés dans
Un répertoire **Themes** global au serveur

A part cette différence, la création et l'utilisation des thèmes se font de la même façon.

ASP.NET & DESIGN – MISE EN PLACE

La première étape consiste à créer le répertoire de thème.

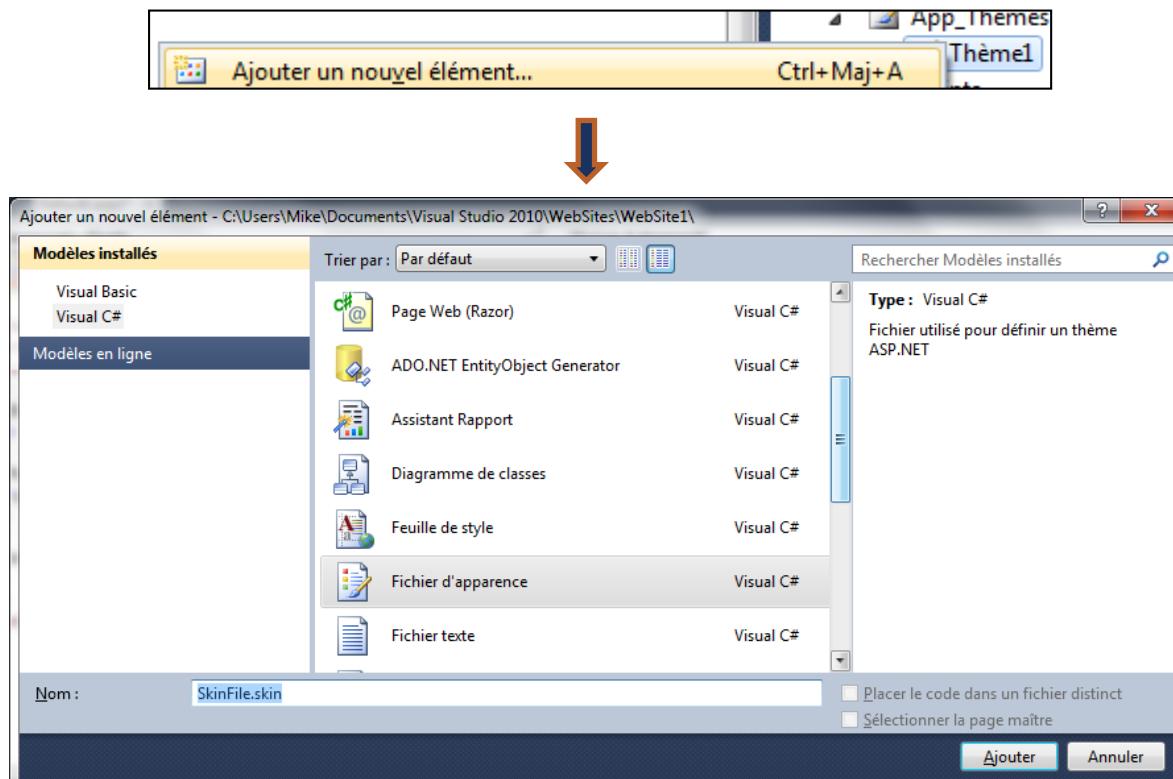


ASP.NET & DESIGN – MISE EN PLACE

Le répertoire **App_Themes** est le répertoire de base permettant de gérer les thèmes.

Lors de la création de ce répertoire, Visual Studio crée par la même occasion le répertoire **Theme1** représentant le thème de départ.

L'étape suivante est la **création du fichier .SKIN**.

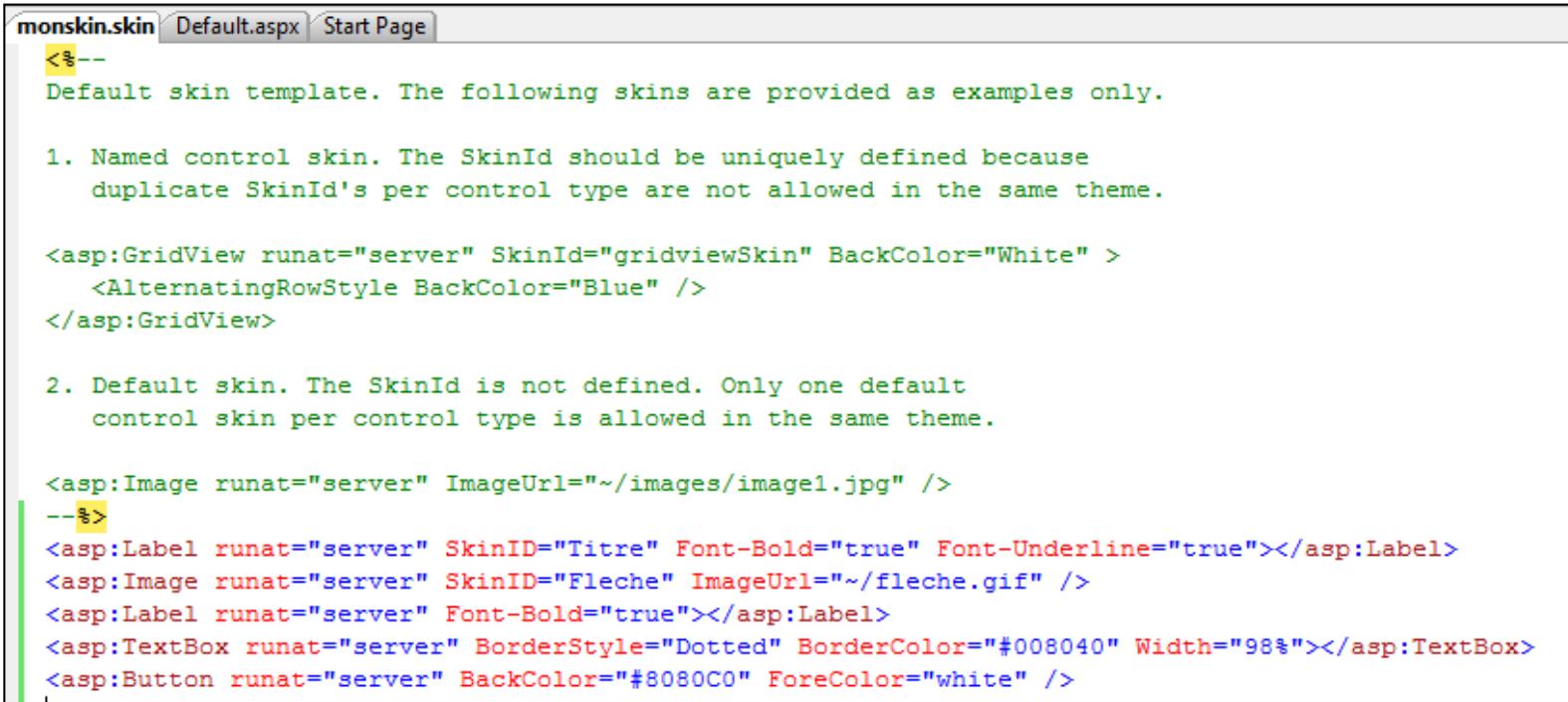


ASP.NET & DESIGN – MISE EN PLACE

Un fichier skin est constitué simplement de déclarations de contrôles avec uniquement les propriétés que l'on désire redéfinir et **sans l'attribut ID**

```
<asp:Button runat="server" BackColor="Red" ForeColor="White" Font-Name="Arial" Font-Size="9px" />
```

Pour créer une apparence, une méthode simple consiste à ajouter le contrôle à une page et à le configurer comme il vous convient.

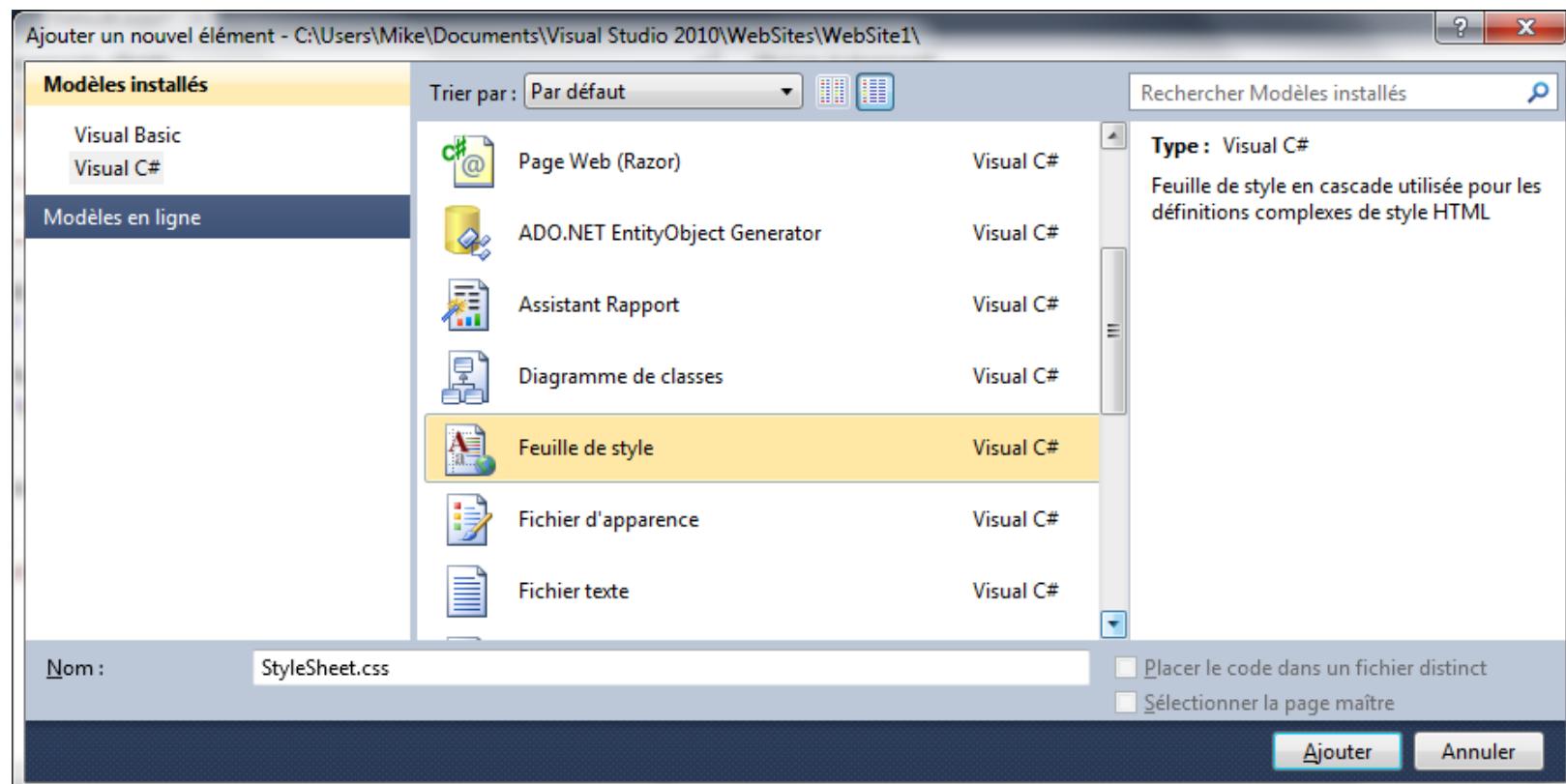


The screenshot shows a browser window with the title bar "monskin.skin" and tabs "Default.aspx" and "Start Page". The main content area displays the following code:

```
<%--  
Default skin template. The following skins are provided as examples only.  
  
1. Named control skin. The SkinId should be uniquely defined because  
duplicate SkinId's per control type are not allowed in the same theme.  
  
<asp:GridView runat="server" SkinId="gridviewSkin" BackColor="White" >  
    <AlternatingRowStyle BackColor="Blue" />  
</asp:GridView>  
  
2. Default skin. The SkinId is not defined. Only one default  
control skin per control type is allowed in the same theme.  
  
<asp:Image runat="server" ImageUrl="~/images/image1.jpg" />  
--%>  
<asp:Label runat="server" SkinID="Titre" Font-Bold="true" Font-Underline="true"></asp:Label>  
<asp:Image runat="server" SkinID="Fleche" ImageUrl="~/fleche.gif" />  
<asp:Label runat="server" Font-Bold="true"></asp:Label>  
<asp:TextBox runat="server" BorderStyle="Dotted" BorderColor="#008040" Width="98%"></asp:TextBox>  
<asp:Button runat="server" BackColor="#8080C0" ForeColor="white" />
```

ASP.NET & DESIGN – MISE EN PLACE

Nous pouvons de la même manière ajouter une feuille de style.



ASP.NET & DESIGN – MISE EN PLACE

The screenshot shows the Visual Studio interface. On the left, the Solution Explorer window displays a project named "LesThemes" with the following structure:

- My Project
- App_Themes
 - montheme
 - monskin.skin
 - monStyle.css
 - Default.aspx
 - Web.config

On the right, there is a code editor window titled "monStyle.css*". It contains the following CSS code:

```
body
{
    background-color: #E8E8E8;
    font-family: Verdana;
    font-size: 12px;
}

table
{
    border: 2px solid #CCC;
}
```

Bien entendu, tout ce qui compose une feuille de style est permis dans cette feuille obtenue

ASP.NET & DESIGN – MISE EN PLACE

La dernière étape consiste à appliquer le thème à une page.

Pour cela, il nous faut assigner la propriété **Theme** de la balise *Page* de la page web avec le nom du thème désiré.

```
<%@ Page Title="Page d'accueil" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
    CodeFile="Default.aspx.cs" Inherits="_Default" Theme="Thème1" %>
<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
```

Une fois le thème associé à la page, il est possible de redéfinir les contrôles afin que ceux-ci reflètent le style défini grâce à l'attribut **SKINID**

```
<td colspan="2">
    <asp:Image ID="Image_Fleche" SkinID="Fleche" runat="server" />&nbsp;
    <asp:Label ID="Label_Titre" SkinID="Titre" runat="server" Text="Formulaire d'exemple">
</td>
:>
```

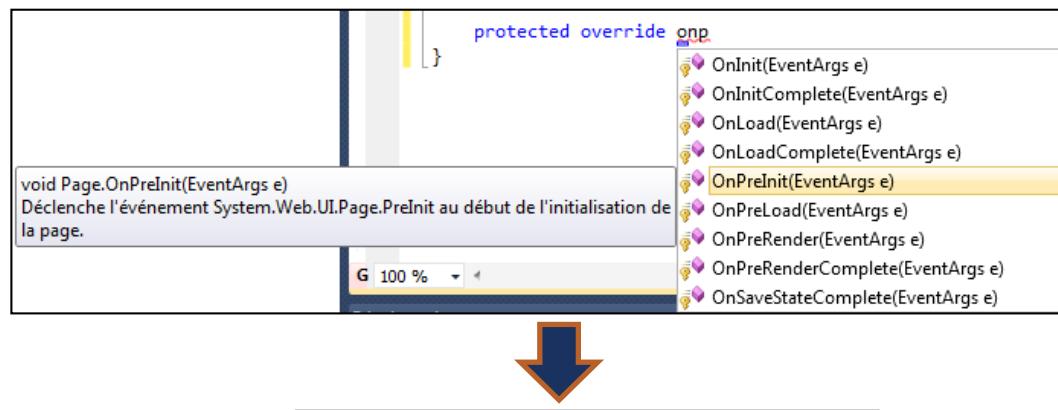
Il est tout à fait possible de modifier le thème durant l'exécution de l'application Web.

Pour ce faire, il suffit tout simplement de setter la propriété *Theme* de votre objet *Page* avec le nom du thème à utiliser.

Ceci ne peut être fait qu'au moment de l'évènement PreInit ou alors avant cet évènement

ASP.NET & DESIGN – MISE EN PLACE

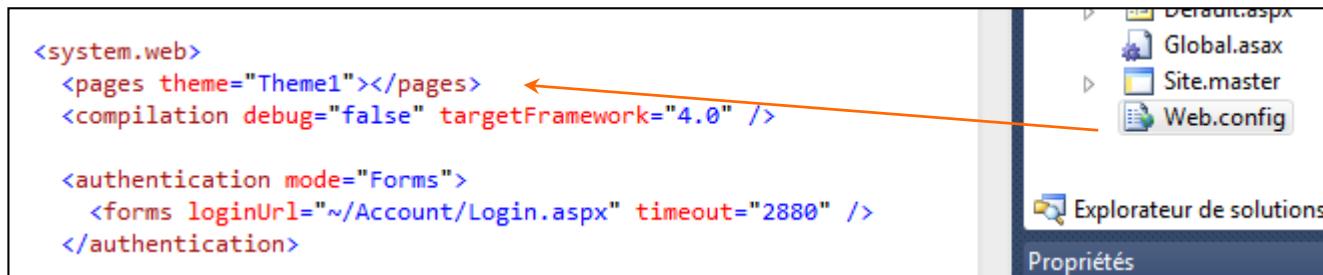
Si nous désirions par exemple changer le style de la page suivant le click sur un bouton,
Il nous faut redéfinir cette méthode **OnPreinit**



```
protected override void OnPreInit(EventArgs e)
{
    if (Request.QueryString["Theme1"] != null)
    {
        Page.Theme = "Theme1";
    }
    if (Request.QueryString["Theme2"] != null)
    {
        Page.Theme = "Theme2";
    }
    if (Request.QueryString["Theme3"] != null)
    {
        Page.Theme = "Theme3";
    }
}
```

ASP.NET & DESIGN – MISE EN PLACE

Pour appliquer un thème pour tout le site, il faut modifier le fichier **web.config**



Plus besoin de préciser le thème pour chaque thème via la propriété Theme de la balise page de vos pages aspx.

ASP.NET & DESIGN – MISE EN PLACE

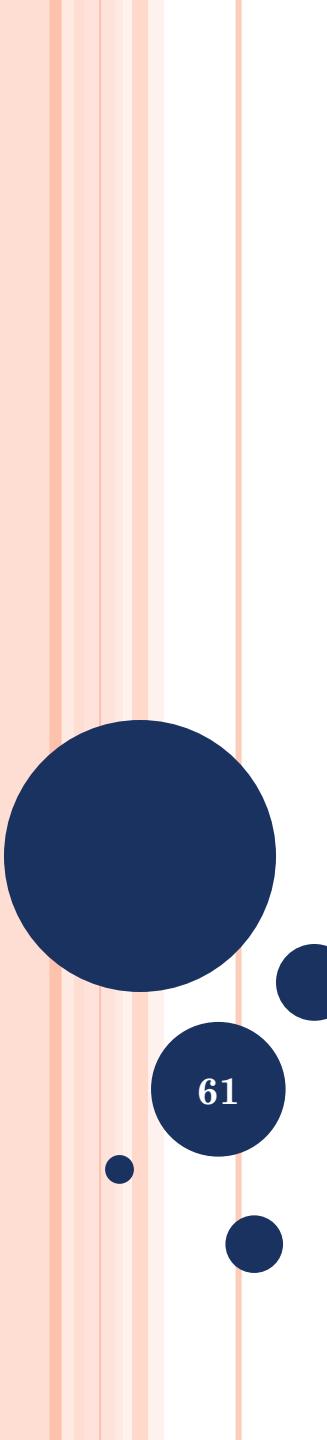
- Exercice

Compléter l'exercice des master pages afin qu'un thème mette en forme les zones de textes, les labels, un titre, les messages d'erreur et les formulaires en général.

Pour cet exercice, utilisez non seulement les feuilles de style mais aussi les fichiers .SKIN.

Le design a peut d'importance, c'est le principe qui prime cependant afin de vous aider, voici un print screen qui peut-être obtenu en utilisant les techniques de themes

The screenshot shows a web page titled "Asp.Net Pratique". At the top, there is a navigation bar with a dark background and white text. Below it, a section titled "Exemples de validations" contains several input fields for "Nom", "Prenom", "Pseudo", "Retaper le pseudo", "Année de naissance", and "Adresse e-mail". To the right of these fields is a Microsoft ASP.net logo. Below this section is a "CustomValidator" control with the message "Entrez un nombre pair". At the bottom of the page is a red-bordered box labeled "Messages d'erreur".



ASP.NET – WEB PARTS & CUSTOM CONTROLS

Michaël Person – Cognitic Sprl

61

CRÉER SES PROPRES CONTRÔLES

- 3 sortes de contrôles
 - Custom Controls
 - User Controls
 - WebParts

Ces trois sortes de contrôles permettent créer des composants réutilisable dans nos pages ASP.NET.

- **Les users controls**

Les « users controls » ne sont pas difficile à créer :

Si vous connaissez l'ASP.NET, vous savez faire des users controls.

En effet, un web user contrôle est un « simple » conteneur dans lequel vous écrivez votre code ASP.NET, HTML...

Ensuite, un simple glissé déposer permet d'obtenir le visuel et le fonctionnel de celui-ci...

CRÉER SES PROPRES CONTRÔLES

The screenshot illustrates the creation of a custom User Control named "WUC_Form.ascx". On the left, two forms are shown: "Simple ASP.NET" and "User Control". The "Simple ASP.NET" form contains fields for Name, Street, City, Country (set to France), and State/Province. The "User Control" form contains similar fields but with different values: Name, Street, City, Country (set to Belaiaue), and State/Province (set to Bruxelles). Both forms have Save and Delete buttons. On the right, the ASPX page code is displayed, showing the registration of the User Control and its usage within the page structure.

```
<%@ Page Title="Page d'accueil" Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="UserControl_Exemple_1._Default" %>
<%@ Register src="WUC_Form.ascx" tagname="WUC_Form" tagprefix="uc1" %>

<html>
  <head>
    <title>Exemple 1 - User Control</title>
  </head>
  <body>
    <form runat="server">
      <div>
        <h1>Simple ASP.NET</h1>
        <table>
          <tr>...</tr>
          <tr>...</tr>
          <tr>...</tr>
          <tr>...</tr>
          <tr>...</tr>
          <tr>...</tr>
        </table>
        <h1>User Control</h1>
        <uc1:WUC_Form ID="WUC_Form1" runat="server" />
      </div>
    </form>
  </body>
</html>
```

CRÉER SES PROPRES CONTRÔLES

Cette solution, si elle l'avantage d'être simple, nous impose quelques limitations.

La limitation majeur est qu'un « user control » ne peut pas hériter des autres contrôles ASP.NET.

La seconde limitation c'est la réutilisation.

- les « users controls » ne peuvent pas être mis en commun entre différents projets ou sites web. Ils sont utilisables uniquement dans le projet où ils sont créés

Il n'y a pas moyen de contourner cette limitation

CRÉER SES PROPRES CONTRÔLES

- **Les customs controls**

Pour la création des customs controls, pas d'interface visuel de création, uniquement la partie code est disponible.

Si on désire ajouter une instance d'un nouveau contrôle, il faut donc créer celui-ci par code et l'ajouter à la collection du Custom control.(Pas de facilité Drag&Drop)

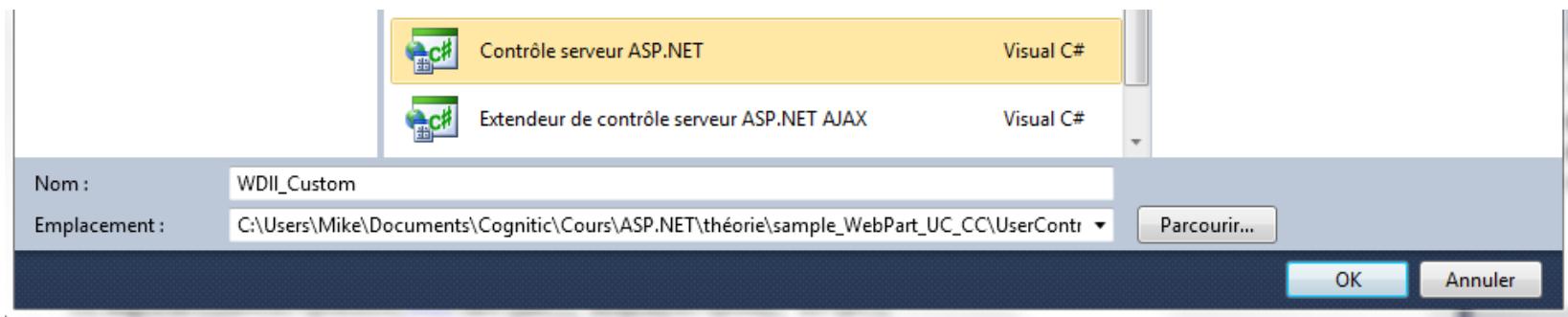
Les customs controls peuvent hériter des autres contrôles et les étendre.

Ils peuvent également être inclus dans une bibliothèque de webuser control et donc être réutilisé dans différents projets

CRÉER SES PROPRES CONTRÔLES

- Créer un lien rollover grâce à un custom control

Il faut tout d'abord créer un projet de type Contrôle serveur ASP.NET



CRÉER SES PROPRES CONTRÔLES

- Le custom control

```
[DefaultProperty("Text")]
[ToolboxData("<{0}:ServerControl1 runat=server>/<{0}:ServerControl1>")]
public class RollOverButton : LinkButton
{
    [Bindable(true),
     Category("Appearance"),
     DefaultValue("")]
    public string ImageUrl
    {
        get
        {
            object o = ViewState["RollOverImageUrl"];
            if (o != null)
                return (string)o;
            else
                return String.Empty;
        }
        set
        {
            ViewState["RollOverImageUrl"] = value;
        }
    }
}
```

CRÉER SES PROPRES CONTRÔLES

- Le custom control

```
[Bindable(true),  
Category("Appearance"),  
DefaultValue("")]  
public string RollOverImageUrl  
{  
    get  
    {  
        object o = ViewState["RollOverRollOverImageUrl"];  
        if (o != null)  
            return (string) o;  
        else  
            return String.Empty;  
    }  
  
    set  
    {  
        ViewState["RollOverRollOverImageUrl"] = value;  
    }  
}
```

```
protected override void RenderContents(HtmlTextWriter output)  
{  
    if (this.ImageUrl != String.Empty)  
    {  
        Image myImage = new Image();  
        myImage.Attributes["name"] = this.ClientID + "_img";  
        myImage.ImageUrl = this.ImageUrl;  
        myImage.AlternateText = this.Text;  
        myImage.Height = this.Height;  
        myImage.Width = this.Width;  
        myImage.ToolTip = this.ToolTip;  
        myImage.RenderControl(output);  
    }  
    else  
        base.RenderContents(output);  
}
```

CRÉER SES PROPRES CONTRÔLES

- Le custom control

```
protected override void OnPreRender(EventArgs e)
{
    ClientScriptManager CS = Page.ClientScript;
    string imageLoadScriptKey = "rolloverImageBuildingCode" + this.ClientID;
    string script =
        @"<script language=""JavaScript"""
        <!--
            " + this.ClientID + "_img_out = new Image(); " +
            this.ClientID + @_img_out.src = """" + this.ImageUrl + @"""";
            " + this.ClientID + "_img_over = new Image(); " +
            this.ClientID + @_img_over.src = """" + this.RollOverImageUrl + @"""";
        // -->
    </script>";
    CS.RegisterClientScriptBlock(this.GetType(), imageLoadScriptKey, script);

    this.Attributes["onmouseover"] =
        "rolloverServerControl_display('" + this.ClientID +
            "_img', " + this.ClientID + "_img_over);";
    this.Attributes["onmouseout"] =
        "rolloverServerControl_display('" + this.ClientID +
            "_img', " + this.ClientID + "_img_out);";

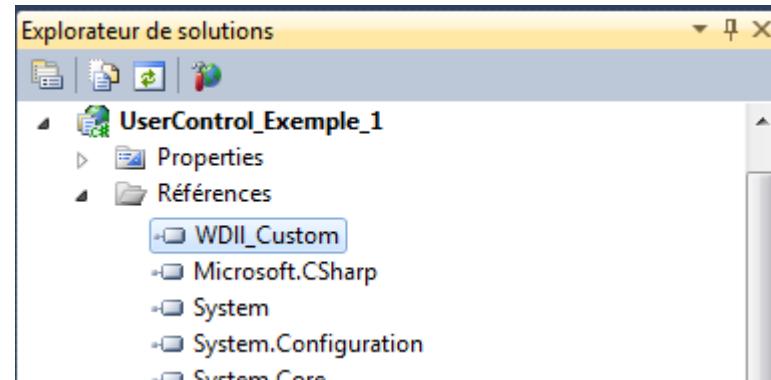
    const string displayScriptKey = "rolloverImageLoadScript";

    if (!CS.IsClientScriptBlockRegistered(displayScriptKey))
    {
        script = @"<script language=""JavaScript"""
        <!--
            function rolloverServerControl_display(imgName, imgUrl) {
                if (document.images && typeof imgUrl != 'undefined')
                    document[imgName].src = imgUrl.src;
            }
        // -->
    </script>";
    CS.RegisterClientScriptBlock(this.GetType(), displayScriptKey, script);
}

base.OnPreRender(e);
}
```

CRÉER SES PROPRES CONTRÔLES

- Utilisation du custom control



```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="UseRolloverBtn.aspx.cs" Inherits="UserControl_Exemple_1.UseRolloverBtn" %>
<%@ Register Assembly="WDll_Custom" Namespace="WDll_Custom" TagPrefix="cc1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body style="background-color:Silver;" >
    <form id="form1" runat="server">
        <div>
            <cc1:RollOverButton runat="server" ID="RollOver1" ImageUrl="images/voirtarif-off.gif" RollOverImageUrl="images/voirtarif-on.gif"></cc1:RollOverButton>
        </div>
    </form>
</body>
</html>
```

CRÉER SES PROPRES CONTRÔLES

- Exercice

Utiliser le service Weather.com

(<https://registration.weather.com/ursa-wow/step1?from=WOWsSalesPage>)

afin de créer un custom control qui affichera la météo sur votre page web suivant les propriétés qui lui seront affectées

CRÉER SES PROPRES CONTRÔLES

- **Les Webparts**

Les webparts apparaissent tout d'abord dans Sharepoint(2003) et ce n'est qu'à partir d'ASP.NET 2.0 qu'ils sont inclus dans le développement ASP.

Sans les webparts, le développeur crée les pages webs et les utilisateurs ne peuvent que les consulter.

Avec les webparts, les utilisateurs ont la possibilité de construire leur page web sans avoir recourt à un développeur

Le développeur construit des composants permettant la personnalisation des pages web par l'utilisateur

CRÉER SES PROPRES CONTRÔLES

- **Les avantages de la création de ses propres contrôles**

- **Productivité**

En créant des contrôles réutilisable, plus besoin de réinventer la roue pour développer des fonctionnalités semblable dans des applications différentes

- **Standardisation**

En utilisant les même composants pour des actions identiques, vous avez la garantie que les fonctionnalités soient implémentées toujours de la même façon

- **Simplification**

En répartissant les fonctionnalités entre des composants spécialisés et d'autres parties de l'application (workflow, business logic, data access,...), la maintenance de chaque partie s'en voit simplifiée.

CRÉER SES PROPRES CONTRÔLES

- **Les avantages supplémentaires des webparts**

- **Intégration**

Les webparts présents sur une même page peuvent échanger des informations

- **Les propriétés peuvent être changer par l'utilisateur**

Durant l'exécution, les utilisateur peuvent customiser les webparts via les propriétés pour de futures consultations de la page.

- **Design**

Les webparts peuvent être ajoutés ou supprimés d'une page ou juste minimisés, maximisés

CRÉER DES PAGES CUSTOMISÉES

- Avant d'envisager les pages customisées, nous nous poserons les questions suivantes :
 - Quel est la charge de travail pour activer la personnalisation?
 - Quels seraient les avantages de la personnalisation pour notre application ?
 - Dois-je écrire énormément de code spécialisé pour nos pages customisées?
 - Qu'est-ce que la personnalisation doit apporter à nos utilisateurs?

CRÉER DES PAGES CUSTOMISÉES

- Le Web part framework control

- WebParManager

- Ce contrôle permet la customisation de la page

- Zones

- Ce contrôle définit les sections de la pages que l'utilisateur peut customiser

- 1. WebPartZone

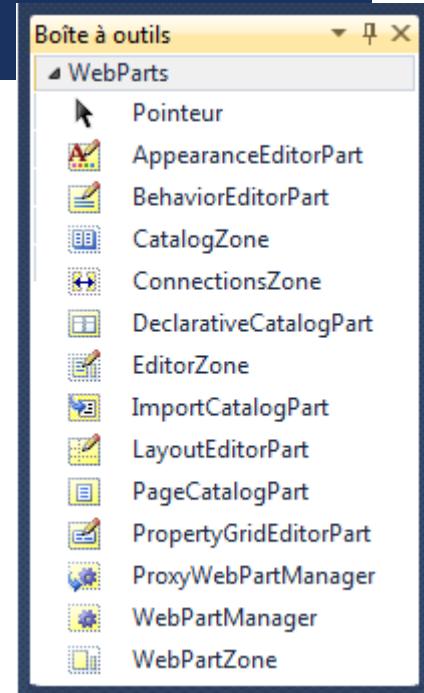
- Contient les contrôles

- 2. CatalogZone

- Contient la liste des contrôles qui peuvent être inclus dans la page

- 3. EditorZone

- Contient le contrôle qui permet de modifier les webparts directement sur la page



CRÉER DES PAGES CUSTOMISÉES

- **Le Web part framework control**

- Editors

Cet outils permet aux utilisateurs de customiser les contrôles

1. AppearanceEditorPart

Contrôle l'apparence du webpart

2. BehaviorEditorPart

Permet à l'utilisateur de modifier les propriétés des webparts

partagés

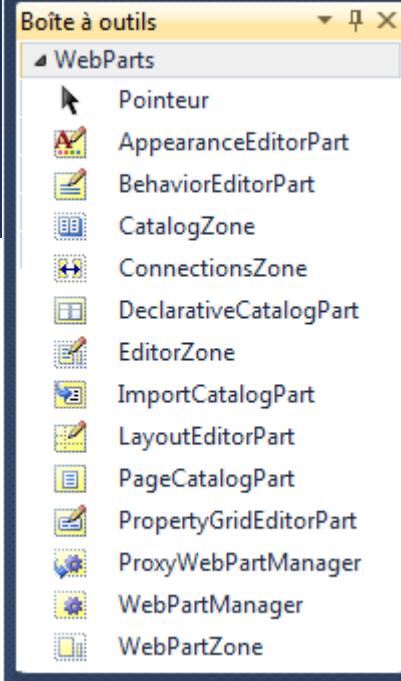
3. LayoutEditorPart

Permet à l'utilisateur de déplacer les webparts entre

les zones

4. PropertyGridEditorPart

Permet à l'utilisateur de changer toutes les propriétés du contrôle personnalisé non héritées



CRÉER DES PAGES CUSTOMISÉES

- **Le Web part framework control**

- Catalogs

Cet outils permet de lister les contrôles qui peuvent être inclus dans la page

- 1. DeclarativeCatalogPart

Liste les contrôles qui ne sont pas encore ajoutés dans la page

- 2. PageCatalogPart

Liste les contrôles qui ont déjà été ajouté dans la page mais qui ne sont pas visible (ceux qui sont fermés)

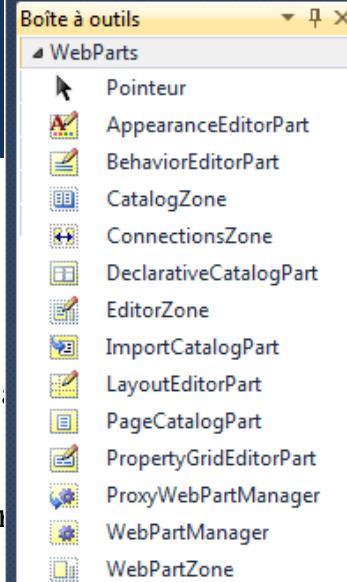
- 3. ImportCatalogPart

Liste les contrôles du serveur qui peuvent être ajouté

après importation

- 4. ConnectionZone

Contient les objets qui permettent l'interconnexion entre les différents contrôles des zones webpart



CRÉER DES PAGES CUSTOMISÉES

- La création d'une page customisée

1. Ajouter un WebPartManager

```
<asp:WebPartManager ID="WebPartManager1" runat="server"></asp:WebPartManager>
```

L'ajout d'un webPartManager , en plus d'activer la customisation, créé une base de données sous SqlServer afin de stocker les informations de personnalisation.

Si vous obtenez l'erreur suivante :

Une erreur liée au réseau ou spécifique à l'instance s'est produite lors de l'établissement d'une connexion à SQL Server. Le serveur est introuvable ou n'est pas accessible. Vérifiez que le nom de l'instance est correct et que SQL Server est configuré pour autoriser les connexions distantes. (provider: SQL Network Interfaces, error: 26 - Erreur lors de la localisation du serveur/de l'instance spécifiés)

Description : Une exception non gérée s'est produite au moment de l'exécution de la requête Web actuelle. Contrôlez la trace de la pile pour plus d'informations sur l'erreur et son origine dans le code.

Erreur lors de la création automatique du fichier de base de données SQLEXPRESS:

La chaîne de connexion utilise un emplacement de base de données dans le répertoire App_Data de l'application pour spécifier une instance SQL Server Express locale. Ayant déterminé qu'elle n'existe pas, le fournisseur a tenté de créer automatiquement la base de données des services d'application. Les conditions de configuration requises suivantes sont nécessaires pour vérifier correctement l'existence de la base de données des services d'application et créer automatiquement cette base de données :

CRÉER DES PAGES CUSTOMISÉES

Les webparts ont besoin d'une base de donnée afin de sauvegarder les customisations des utilisateurs.

Cette base de donnée est créée via la commande Aspnet_Regsql.exe présente dans le dossier **systemroot\Microsoft.NET\Framework\numéroversion** de votre serveur Web.

Si vous n'utilisez pas SQL server Express, le message d'erreur peut continuer à apparaître.

Si c'est le cas, il faut configurer un nouveau *PersonalizationProvider*.

Pour cela, il faut modifier le fichier web.config :

- 1) Ajouter une connexion string pour l'accès à DB

```
<connectionStrings>
  <remove name="LocalSqlServer" />
  <add name="DBConn" connectionString="Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=aspnetdb;Data Source=VOSTRO\SQLTTIC"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

- 1) Ajouter les informations nécessaires aux web parts dans *system.web*

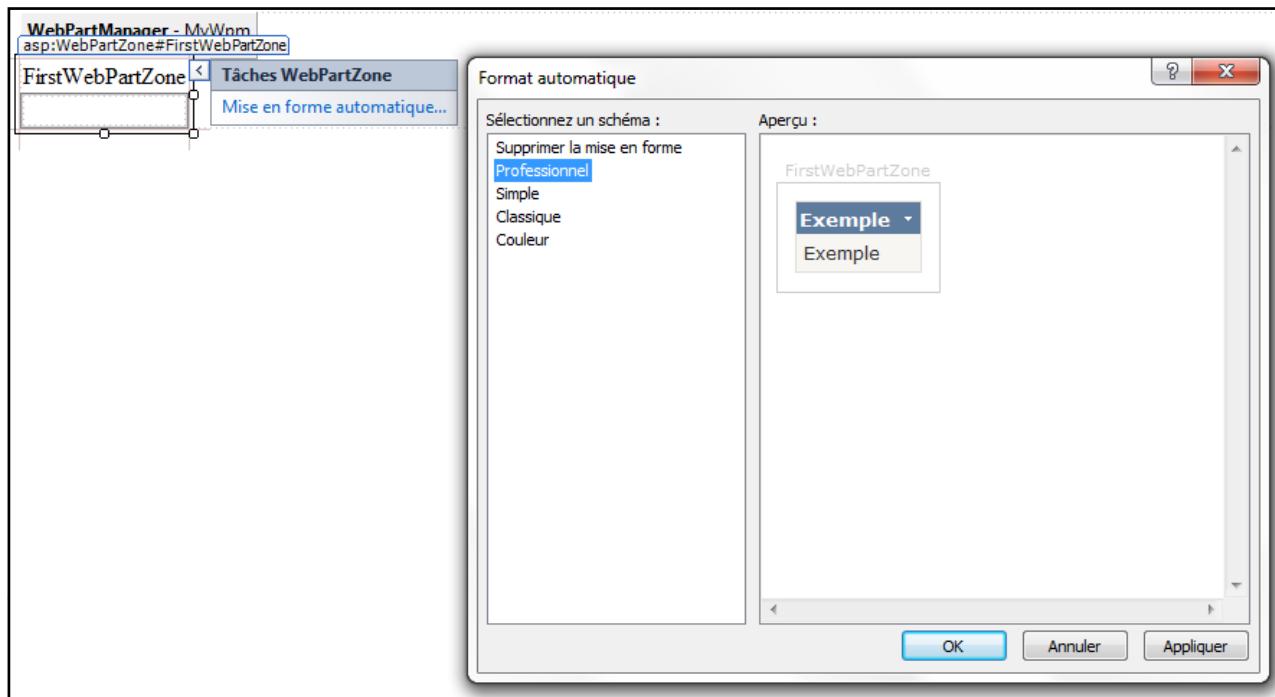
```
<webParts>
  <personalization defaultProvider="AspNetSqlPersonalizationProvider">
    <providers>
      <remove name="AspNetSqlPersonalizationProvider" />
      <add name="AspNetSqlPersonalizationProvider" type="System.Web.UI.WebControls.WebParts.SqlPersonalizationProvider"
        connectionStringName="DBConn" applicationName="/" />
    </providers>
  </personalization>
</webParts>
```

CRÉER DES PAGES CUSTOMISÉES

2. Ajouter une WebPartZone

Afin de pouvoir ajouter les futurs web parts, nous avons besoin d'une WebPartZone

```
<asp:WebPartZone ID="FirstWebPartZone" runat="server"></asp:WebPartZone>
```



CRÉER DES PAGES CUSTOMISÉES

2. Ajouter des contrôles dans les WebPartZones

La partie la plus simple puisqu'elle consiste à simplement construire nos zones comme autant de mini-pages... Du moins c'est ce que l'on pourrait croire.

Malheureusement, si nous désirons simplement glisser-déposer des contrôles afin de construire un web part de login, voici ce que nous obtenons

```
<asp:WebPartZone ID="FirstWebPartZone" runat="server" BorderColor="#CCCCCC"
    Font-Names="Verdana" Padding="6">
    <ZoneTemplate>
        <asp:Label ID="Label1" runat="server" Text="Login"></asp:Label>&nbsp;<asp:TextBox
            ID="txtLogin" runat="server"></asp:TextBox><br />
        <asp:Label ID="Label2" runat="server" Text="Password"></asp:Label>&nbsp;<asp:TextBox
            ID="txtPassword" runat="server"></asp:TextBox><br />
        <asp:Button ID="btnConnect" runat="server" Text="Connectez-vous" />
    </ZoneTemplate>
</asp:WebPartZone>
```

CRÉER DES PAGES CUSTOMISÉES

Pour obtenir quelque chose de correct, nous devrons donc passer par un WebUserControl

Sans titre [6] Réduire Fermer

Login

Password

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="WUC_Login.ascx.cs" Inherits="WebPartFrameworkControl.WUC_Login" %>

<table>
  <tr>
    <td><asp:Label ID="Label1" runat="server" Text="Login"></asp:Label></td>
    <td><asp:TextBox ID="txtLogin" runat="server"></asp:TextBox></td>
  </tr>
  <tr>
    <td><asp:Label ID="Label2" runat="server" Text="Password"></asp:Label></td>
    <td><asp:TextBox ID="txtPassword" runat="server"></asp:TextBox></td>
  </tr>
  <tr>
    <td colspan=2 align=right><asp:Button ID="btnConnect" runat="server" Text="Connectez-vous" /></td>
  </tr>
</table>
```

`this.SecondWebPartZone.WebParts[0].Title = "Authentication";`

Remarque :

COGNITIC Pour changer le titre de la zone

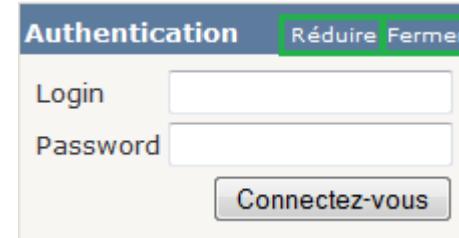
pour ne pas autoriser le déplacement d'un web part entre les zones, il suffit de mettre sa propriété *AllowLayoutChange* à false.

CRÉER DES PAGES CUSTOMISÉES

- **Configuration des *verbs**

Nous pouvons distinguer :

- CloseVerb (inclus par défaut dans le menu)
- ConnectVerb
- DeleteVerb
- EditVerb
- ExportVerb
- HelpVerb
- MinimizeVerb (inclus par défaut dans le menu)
- RestoreVerb



Ils représentent les actions de customisation possible par l'utilisateur de notre page web.

CRÉER DES PAGES CUSTOMISÉES

- **Activer la customisation**

Par défaut l'affichage de la page ne permet pas la personnalisation.

Il faut donc activer celle-ci et mettre en place les composants pour personnaliser la page

La première étape est de configurer le *displayMode* du WebPartManager.

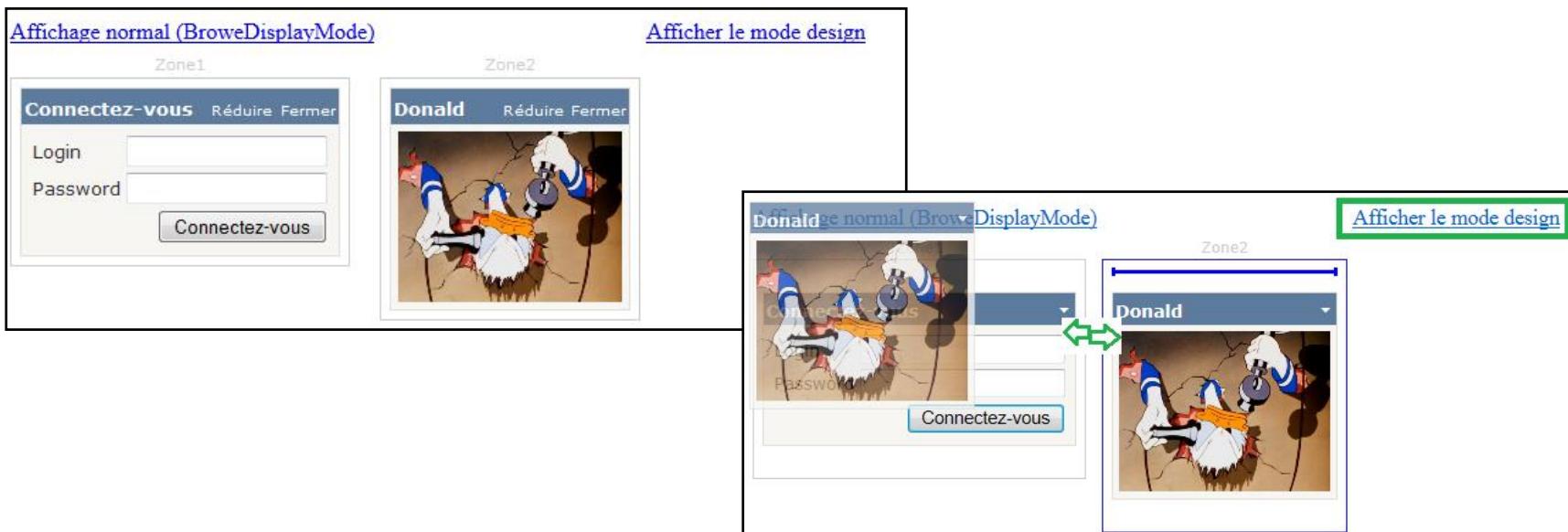
- BrowseDisplayMode (par défaut)
- DesignDisplayMode
- CatalogDisplayMode
- EditDisplayMode
- ConnectDisplayMode

CRÉER DES PAGES CUSTOMISÉES

- DesignDisplayMode

```
WPM_Catalog.DisplayMode = WebPartManager.DesignDisplayMode;
```

Pour permettre le déplacement des webpart entre les zones, nous devons faire passer le *DisplayMode* en *DesignDisplayMode*.

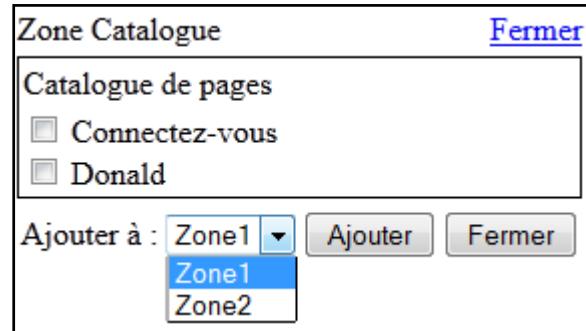


CRÉER DES PAGES CUSTOMISÉES

- CatalogDisplayMode

Pour pouvoir afficher le(s) catalogue(s), nous devons tout d'abord ajouter une *CatalogZone* et ensuite un *PageCatalogPart*.

```
<asp:CatalogZone ID="CatalogZone1" runat="server">
    <ZoneTemplate>
        <asp:PageCatalogPart ID="PageCatalogPart1" runat="server" />
    </ZoneTemplate>
</asp:CatalogZone></td>
```



```
WPM_Catalog.DisplayMode = WebPartManager.CatalogDisplayMode;
```

Lorsque nous exécutons ce code, notre page est vide tant que nous n'avons pas changer le *DisplayMode*

CRÉER DES PAGES CUSTOMISÉES

- EditDisplayMode `<asp:EditorZone ID="EditorZone1" runat="server"> </asp:EditorZone>`

Il existe plusieurs composants pour éditer les propriétés d'un webpart qui doivent être inclus dans un EditorZone

- AppearanceEditorPart

L'utilisateur va pouvoir agir sur certaines propriétés du contrôle dans le webpart tel sa largeur, sa hauteur, son titre, etc...

`<asp:AppearanceEditorPart ID="AppearanceEditorPart1" runat="server" />`



LE VIEWSTATE

ASP.NET = Client-Server

Problème : maintenir l'état des pages suite à « l'aller-retour » serveur.

Les principes pour maintenir l'état des pages :

- Viewstate (niveau page)
- Session (niveau user)
- Application (niveau application)

Tous peuvent être utilisé avec les webparts mais l'utilisation du viewstate apporte des réponses à certains problème liés à l'utilisation des webparts

LE VIEWSTATE

Il y a 3 mécanismes pour enregistrer les données dans le viewState

- **Accès direct**

Les données sont stockées sous forme de Key/value. C'est l'utilisation la plus facile mais la moins efficace, souple

- **Surcharger les méthodes de contrôle**

Ce mécanisme permet de stocker/récupérer de larges quantités de données stockées dans le viewState de façon efficace

- **Utiliser l'espace réservé aux contrôles**

Ce mécanisme permet de séparer les données que nous allons gérer via nos fonctions pour nos contrôles et les données qui vont devoir être gérées par le développeur

LE VIEWSTATE

1. Accès direct

```
ViewState["Info"] = txtinfo.Text;
```

```
if(ViewState["Info"]!=null) lblInfo.InnerText = "Information dans le viewState :" +ViewState["Info"].ToString();
```

Remarque :

Si on garni le viewstate très tôt dans le cycle de vie du contrôle ,dans l'événement init par exemple, il est possible que le mécanisme de chargement de la page efface ces données.

Pour éviter cela :

```
this.TrackViewState();
ViewState["Info"] = txtinfo.Text;
```

Il faut également s'assurer que le ViewState est activé pour le contrôle

```
if (this.IsViewStateEnabled)
{
    this.TrackViewState();
    ViewState["Info"] = txtinfo.Text;
}
```

LE VIEWSTATE

2. Surcharge

Problèmes :

- 1) Utiliser le viewState peut transmettre énormément de données au browser
- 2) Si on a beaucoup de données, il faut mettre en place énormément des paires Key/value

Solution :

Créer un objet personnalisé qui va se charger de gérer la collection de données du viewState .

Pour cela, nous allons surcharger

1) SaveViewState

Tout ce qui est renvoyé par cette fonction est sauvé dans le viewState

2) LoadViewState

Une copie des données sauvees dans saveViewState est passé dans un seul paramètre à cette méthode

LE VIEWSTATE

```
[Serializable()]
public struct StateData
{
    public string strCustNumber;
    public int intCustStatus;
}

StateData sData;
protected override object SaveViewState()
{
    sData.intCustStatus = 9;
    sData.strCustNumber = "A49K72";
    return sData;
}
protected override void LoadViewState(object savedState)
{
    sData = (StateData) savedState;
}

protected void Page_Load(object sender, EventArgs e)
{
    Response.Write(sData.strCustNumber);
}

protected override void OnPreInit(EventArgs e)
{
    if (this.IsTrackingViewState == false)
    {
        this.TrackViewState();
    }
}
```

LE VIEWSTATE

Quand ces méthodes sont-elles appelées ?



- Prerender
- LoadControlState
- LoadViewState**
- CreateChildControls
- SaveControlState
- SaveViewState**
- Render



LE VIEWSTATE

Remarque:

- 1) Tout ce qui est mis dans le viewState n'est pas nécessairement stockée dans le viewState. Seul les objets mis à jour sont stockés afin de maximiser les performances.
- 2) Le suivi des changements dans le viewState n'est pas activé automatiquement dans le début du cycle de vie de la page.
→ Il faut donc TOUJOURS vérifier si le tracking est activé avant de sauver quoi que ce soit dans le viewState si nous désirons le faire dans le INIT, PREINIT,etc...

```
protected override void OnPreInit(EventArgs e)
{
    if (this.IsTrackingViewState == false)
    {
        this.TrackViewState();
        this.ViewState["StateData"] = "essai";
    }
}
```

LE VIEWSTATE

Si le principe démontré jusqu'ici permet de contrôler le viewState pour la page, nous ne pouvons garantir l'utilisation du viewState pour les « user Controls» la composant.

3. L'espace réservé pour les contrôles (ControlState)

Pour pouvoir utiliser le principe du ControlState, il faut surcharger 2 méthodes

- LoadControlState
- SaveControlState

Ces 2 méthodes sont équivalentes aux 2 discutées précédemment.

Pour pouvoir utiliser le ControlState, nous devons appeler *RegisterRequiresControlState*

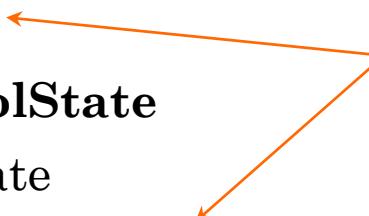
```
private void WebCustomControl1_Init(object sender, System.EventArgs e)
{
    this.Page.RegisterRequiresControlState(this);
}
```

LE VIEWSTATE

Quand ces méthodes sont-elles appelées ?



Prerender
LoadControlState
LoadViewState
CreateChildControls
SaveControlState
SaveViewState
Render



LE VIEWSTATE

```

protected override object SaveControlState()
{
    sData.intCustStatus = 9;
    sData.strCustNumber = "A49K72";
    return sData;
}
protected override void LoadControlState(object savedState)
{
    sData = (StateData) savedState;
}

```

Nettoyer les States

Ça ne peut se faire qu'après les loadState.

Le faire avant ne provoque pas d'erreur mais rien n'est nettoyé

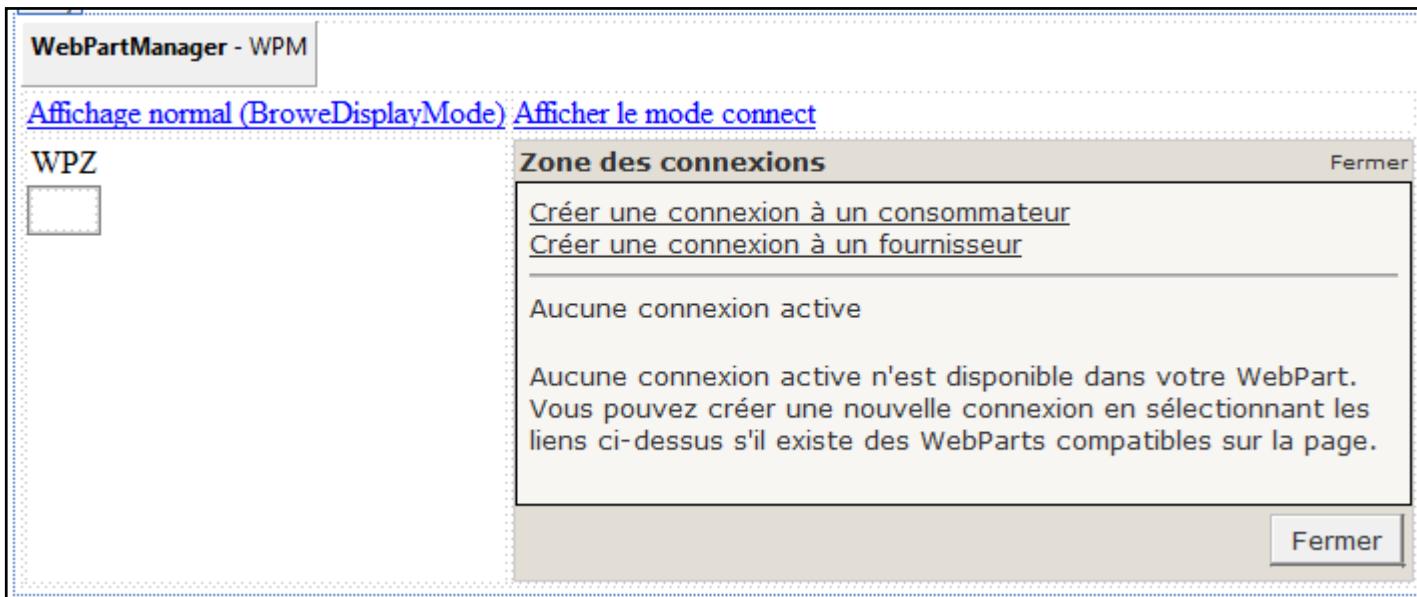
```

protected override void Render(HtmlTextWriter writer)
{
    if (this.HasChildViewState == true && this.IsChildControlStateCleared == false)
    {
        this.ClearChildState();
    }
    else if (this.HasChildViewState == true)
    {
        this.ClearChildViewState();
    }
    else if (this.IsChildControlStateCleared == false)
    {
        this.ClearChildControlState();
    }
    base.Render(writer);
}

```

LES CONNEXIONS ENTRE WEBPARTS

- Pour permettre les connexions entre webparts, il faut d'abord mettre en place
 - Un Web Part Manager
 - Une zone Web Part
 - Une ConnectionZone Web Part



LES CONNEXIONS ENTRE WEBPARTS

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:WebPartManager ID="WPM" runat="server">
            </asp:WebPartManager>
            <table>
                <tr>
                    <td>
                        <asp:LinkButton ID="lkBrowse" runat="server" OnClick="lkBrowse_Click">Affichage normal (BroweDisplayMode)</asp:LinkButton>
                    </td>
                    <td>
                        <asp:LinkButton ID="LkConnect" runat="server" OnClick="LkConnect_Click">Afficher le mode connect</asp:LinkButton>
                    </td>
                </tr>
                <tr>
                    <td valign="top">
                        <asp:WebPartZone ID="WPZ" runat="server">
                        </asp:WebPartZone>
                    </td>
                    <td valign="top">
                        <asp:ConnectionsZone ID="ConnectionsZone1" runat="server" Width="450px">
                        </asp:ConnectionsZone>
                    </td>
                </tr>
            </table>
        </div>
    </form>
</body>
</html>
```

LES CONNEXIONS ENTRE WEBPARTS

- Création des Web Parts connectable
 - Provider WebPart
 - 1) Définir un interface qui va spécifier les données qui vont êtres passées entre les webparts
 - 2) Créer le webpart qui va fournir les données (méthodes et propriétés)
 - 3) Ecrire la routine qui va capturer la connexion au provider
- L'interface

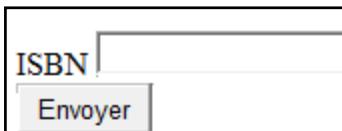
Cet interface permet de définir les données qui peuvent être passées entre les webparts

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

namespace _Connectable
{
    public interface IBookInfo
    {
        string ISBN
        {
            get;
            set;
        }
    }
}
```

LES CONNEXIONS ENTRE WEBPARTS

- Le provider Web Part
 - 1) Implémenter l'interface



```
ISBN   
Envoyer
```

```
public partial class ProviderPart : System.Web.UI.UserControl, IBookInfo
{
    string IBookInfo.ISBN
    {
        get
        {
            return txtIsbn.Text;
        }
        set
        {
            txtIsbn.Text = value;
        }
    }

    protected void Page_Load(object sender, EventArgs e)
    {
    }
}
```

LES CONNEXIONS ENTRE WEBPARTS

2) Créer un point de connexion

```
[System.Web.UI.WebControls.WebParts.ConnectionProvider("Provides ISBN", "ISBNProvider")]
public IBookInfo IBookInfoProvider()
{
    return this;
}
```

Vous pouvez lui donner le nom que vous voulez mais l'en-tête *ConnectionProvider* est obligatoire

```
public partial class ProviderPart : System.Web.UI.UserControl, IBookInfo
{
    string IBookInfo.ISBN
    {
        get
        { return txtIsbn.Text; }
        set
        { txtIsbn.Text = value; }
    }

    [System.Web.UI.WebControls.WebParts.ConnectionProvider("Provides ISBN", "ISBNProvider")]
    public IBookInfo IBookInfoProvider()
    {
        return this;
    }
}
```

LES CONNEXIONS ENTRE WEBPARTS

- Le Consumer Web Part

L'unique obligation est de créer un point de connexion

```
private IBookInfo ibk;  
[System.Web.UI.WebControls.WebParts.ConnectionConsumer("IBookInfo Consumer")]  
public void IBookInfoConsumer(IBookInfo bk)  
{  
    ibk = bk;  
}  
  
protected override void Render(HtmlTextWriter writer)  
{  
    if (ibk == null)  
        lblIsbn.Text = "Pas d'ISBN";  
    else  
        lblIsbn.Text = ibk.ISBN;  
    base.Render(writer);  
}
```

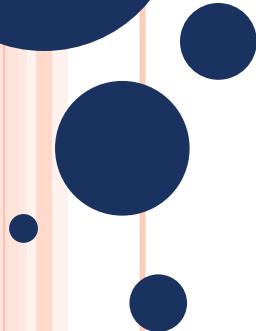
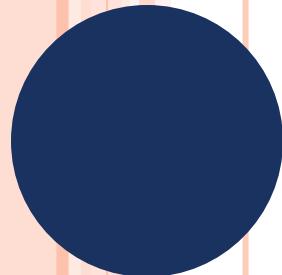
LES CONNEXIONS ENTRE WEBPARTS

[Affichage normal \(BroweDisplayMode\)](#) [Afficher le mode connect](#)

The screenshot shows a SharePoint ribbon interface. On the left, there are two web parts: 'WPZ' (Provider) and 'WPZ2' (Consumer). The 'WPZ' provider has a form with fields for ISBN and an 'Envoyer' button. The 'WPZ2' consumer has a message saying 'L'ISBN choisi : Pas d'ISBN'. A connection dialog is open in the center, titled 'Zone des connexions'. It contains the message 'Envoyer des données à un WebPart' and 'Créez des connexions consommateur pour ce WebPart.' Below this, it says 'Envoyer : Provides ISBN' and 'À :'. A dropdown menu is open, showing 'Connecter' and 'Consumer', with 'Consumer' highlighted. There are 'Fermer' and 'Annuler' buttons at the bottom of the dialog.

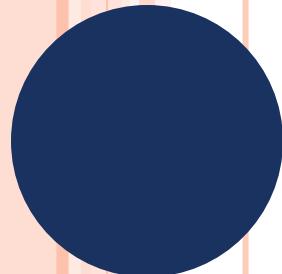
EXERCICE

- Mettre en place un site permettant de s'authentifier.
 - Une fois authentifié, le membre peut customiser sa page à partir d'une librairie de webpart
 - Un webpart affichant la météo
 - un webpart affichant les dernières news d'un flux rss (paramétrable dans le webpart)
 - un webpart permettant d'afficher une boîte gmail connecté à un webpart permettant d'afficher le message
 - un webpart avec recherche google
 - un webpart affichant ses informations



ASP.NET MVC 3

Michael Person
Cognitic Sprl



108



INTRODUCTION



ASP.NET MVC 3 - INTRO

- Au commencement vinrent les Webforms

ASP.NET Web Forms

A set of UI components (pages, buttons, etc.) plus a stateful, object-oriented GUI programming model

ASP.NET

A way to host .NET applications in IIS (Microsoft's web server product), letting you interact with HTTP requests and responses

.NET

A multilanguage-managed code platform
(brand-new at the time—a landmark in its own right)

ASP.NET MVC 3 - INTRO

1. But des webforms

- Cacher HTTP & HTML en modelisant des interfaces via des contrôles serveurs.
- Chaque contrôle sauvegarde son propre état grâce au *ViewState* (sérialisation des données via http Request)
- Chaque contrôle génère son propre HTML permettant l'affichage web de celui-ci et connecte les événements client avec le code serveur correspondant.

1. Les faiblesses des webforms

- Poids du ViewState

Le mécanisme du *ViewState* qui maintient l'état des contrôles entre les dialogues Client/Server entraîne un transfert de larges blocs de données.

- Augmente le temps de réponse du server
- Augmente le besoin de bande passante

- Illusion de la séparation des couches

Le modèle *code-behind* permet de séparer la couche HTML du code applicatif.

Cependant, dans la plupart des cas, de part cette pseudo séparation, le développeur est tenté de mixer le côté *présentation* et la *logique applicative*.

ASP.NET MVC 3 - INTRO

- **Limitation du contrôle du code HTML généré**

L'HTML généré par nos contrôles serveur n'est pas nécessairement l'HTML que l'on souhaite et mettre en page via le css peut s'avérer ardu. De plus, le rendu sur certains navigateur échoue parfois de part le non respect des standards.

- **Testing**

Il est quasiment impossible de tester le design séparément du code applicatif et mettre en place des tests automatiques est un challenge.

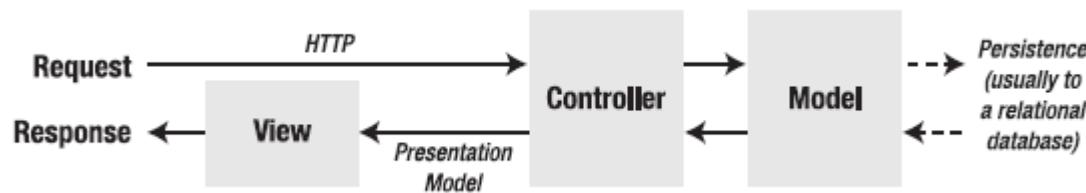
ASP.NET MVC 3 - INTRO

- **Pattern MVC et ASP.NET MVC Framework**

Le pattern MVC n'est pas neuf (1978) mais il a gagné en popularité dans le monde web pour les raisons suivantes :

1. Les interactions utilisateur avec une application MVC un cycle naturel
L'utilisateur effectue une action et l'application répond en présentant une vue mise à jour à celui-ci. Très intéressant pour une application web se basant sur des *request* et *responses*.
2. Les applications web requièrent des combinaisons de plusieurs technologies (DB, HTML, Services,...) généralement décomposées en différentes couches

ASP.NET MVC Framework implémente une version MVC spécialement renommée pour le monde web.



ASP.NET MVC 3 – GET READY

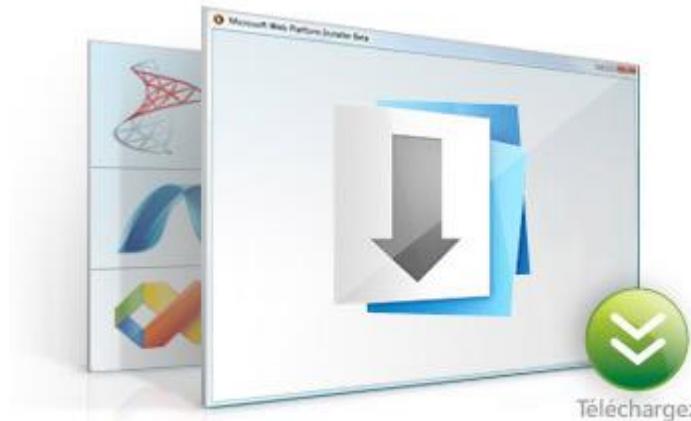
Avant de pouvoir développer une application MVC Framework, quelques manipulations sont nécessaires.

- **Mise à jour de votre Visual Studio 2010**

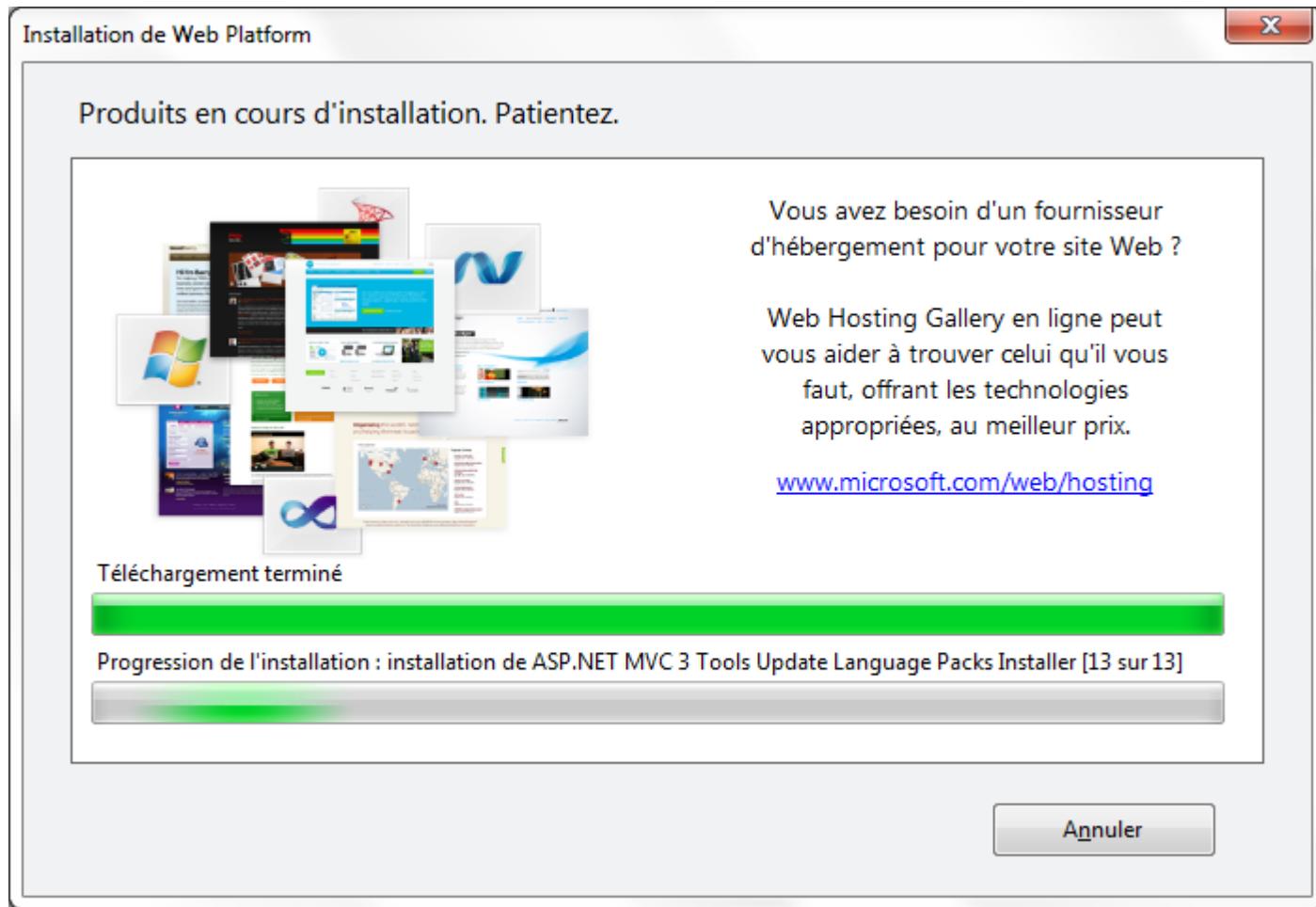
La façon la plus simple de mettre à niveau votre VS2010, incluant tous les composants nécessaires ainsi que MVC, est d'utiliser l'outil gratuit WebPI.

(<http://www.microsoft.com/web/downloads/platform.aspx>)

Cet outil permet de télécharger et installer des composants via la plateforme web Microsoft



ASP.NET MVC 3 – GET READY



ASP.NET MVC 3 – GET READY

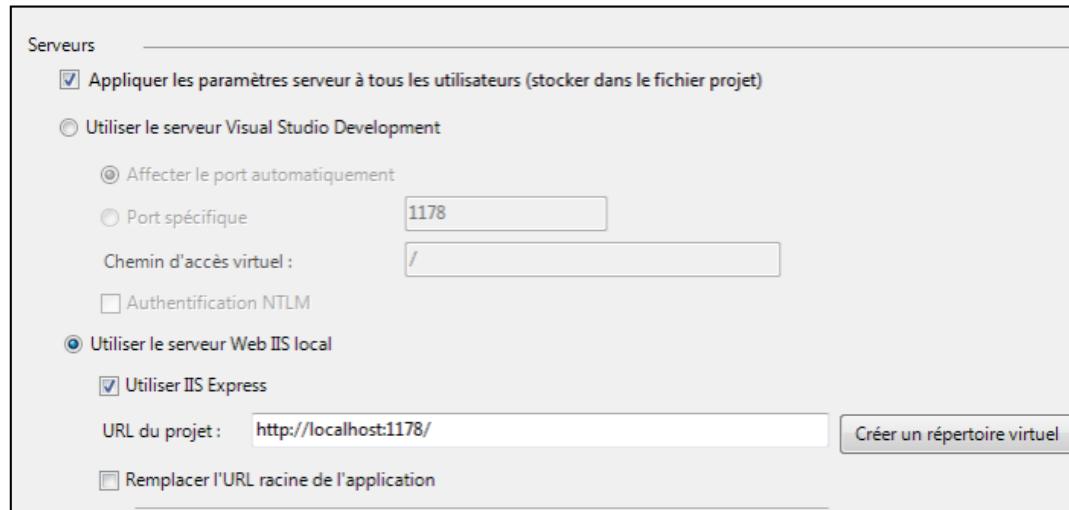
- **IIS**

Visual studio inclus un serveur web mais ce server n'inclus pas toutes les fonctionnalités de IIS (ex: SSL).

Il est donc recommandé d'installer un IIS.

Soit la version Full, mais nous perdons l'intégration avec visual studio.

Soit la version express permettant de bénéficier des l'intégration à VS et des toutes les fonctionnalités de la version full. Il suffit alors de le choisir dans les propriétés web du projet



ASP.NET MVC 3 – GET READY

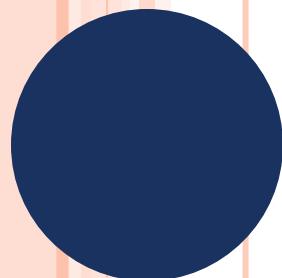
Si vous voulez utiliser la version full de IIS, il vous faut :

- 1) Le framework .NET 4
- 2) Les web deployment tools 2.0

De plus, vous devrez configurer IIS.

Pour un tutoriel complet , rendez-vous sur

<http://learn.iis.net/page.aspx/984/configure-web-deploy>

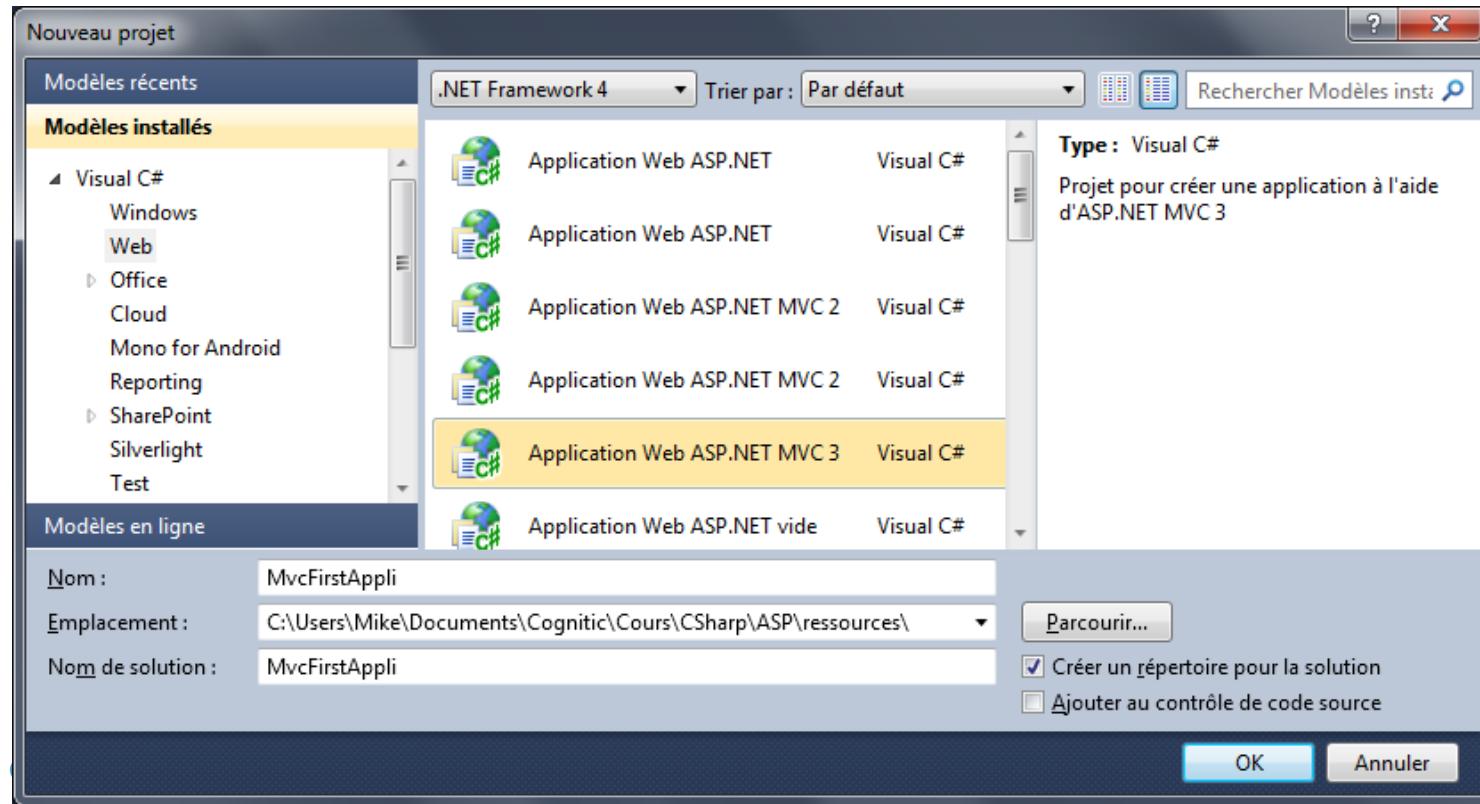


117

PREMIÈRE APPLI

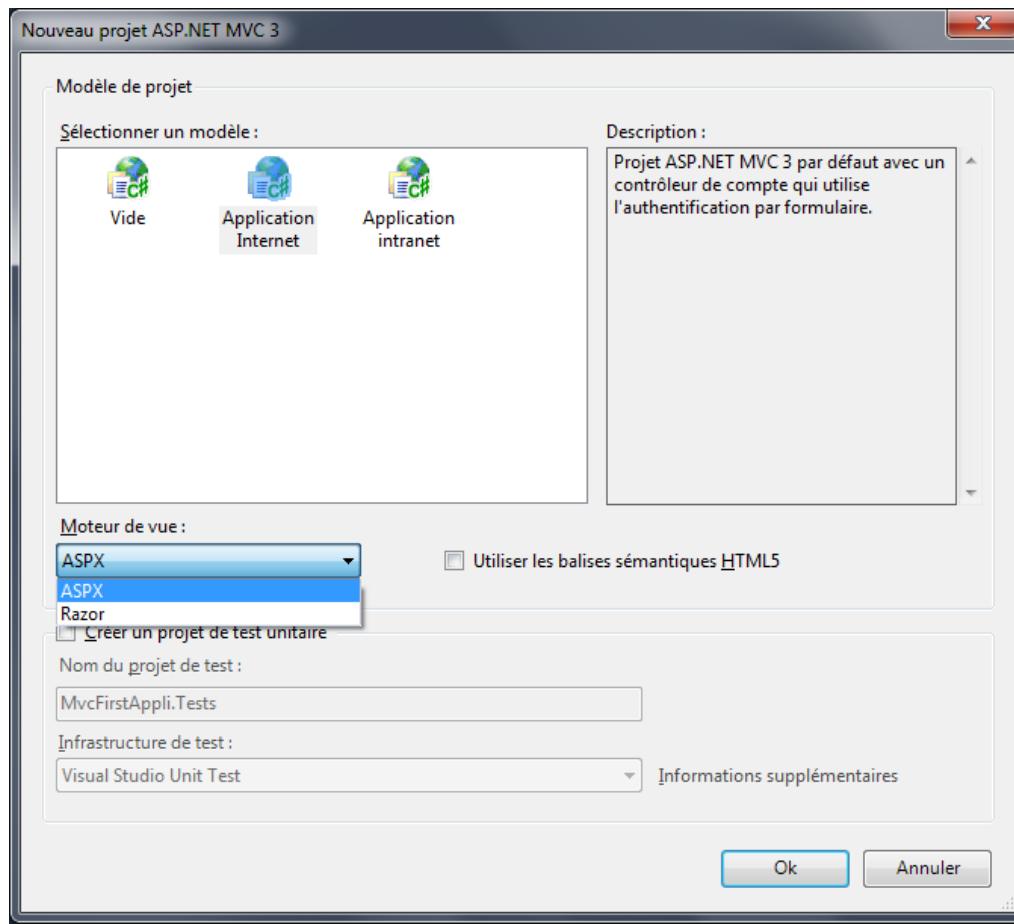
ASP.NET MVC 3 – PREMIÈRE APPLI

Afin d'aborder MVC 3, nous allons créer notre première application.



ASP.NET MVC 3 – PREMIÈRE APPLI

A l'étape suivante, nous avons le choix entre trois templates MVC



ASP.NET MVC 3 – PREMIÈRE APPLI

- **Empty**

Créé une application avec le minimum de composants requis pour MVC 3

- **Internet Application**

Créé un simple exemple incluant l'enregistrement des utilisateur, le login, la navigation et un style visuel que nous pourrons modifier par la suite

- **Intranet Application**

Ressemble au template internet mais construit pour s'intégrer dans un environnement où les utilisateurs s'authentifient via une active directory.

Pour notre premier exemple, nous allons commencer par **Empty** sans checker **HTML5 semantic markup**.

Nous utilisons également le moteur de vue Razor.

Ce moteur de vue remplace les tags ASP habituels (<% %>) par @. Cette nouvelle notation est plus rapide à écrire et est plus rapidement compilée.

Microsoft recommande ce moteur.

ASP.NET MVC 3 – PREMIÈRE APPLI

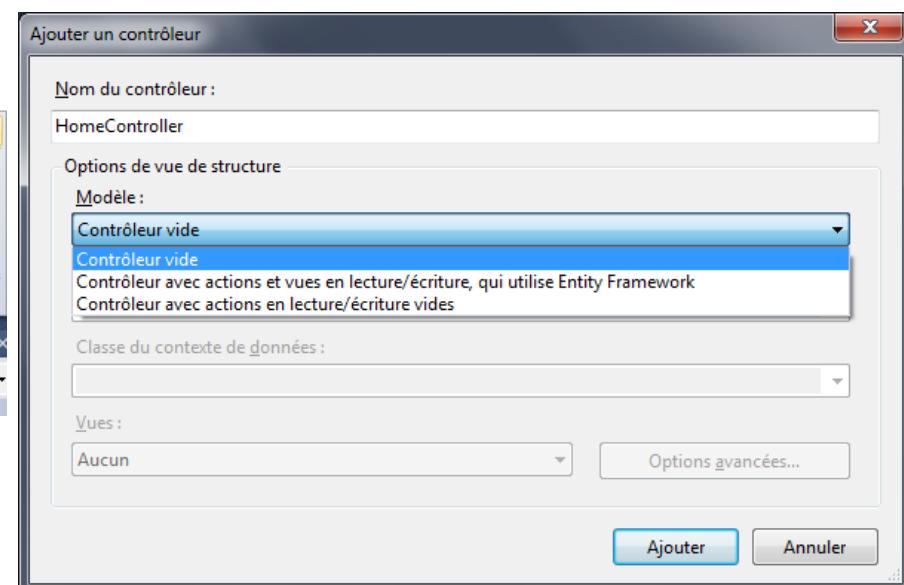
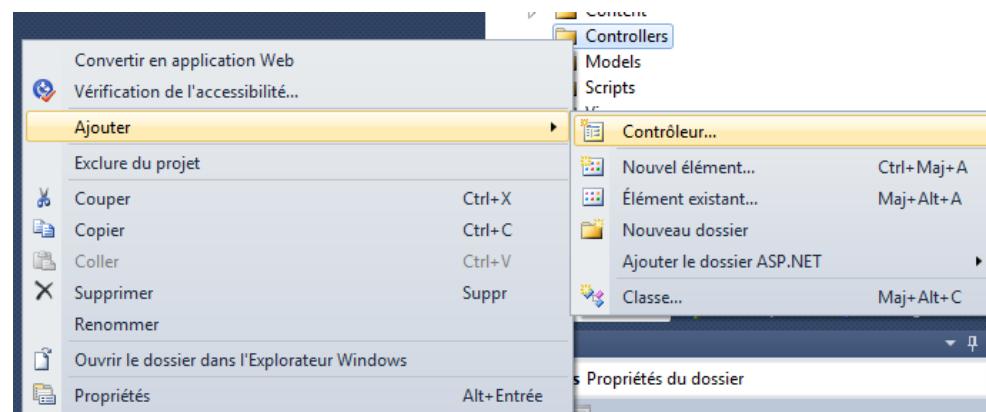
- Ajout de notre premier controller

Dans l'architecture MVC, les requêtes sont gérées par un *Controller*.

Il s'agit en fait d'une simple classe C# héritant de *System.Web.Mvc.Controller*.

Chaque méthode publique est appelée *action method* qui peut être invoquées via une url.

Par convention : le controller est placé dans un dossier Controllers

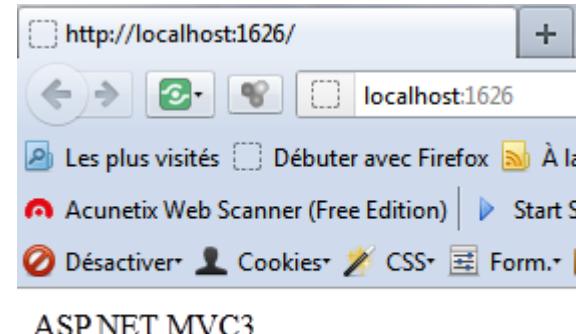


ASP.NET MVC 3 – PREMIÈRE APPLI

Il est possible de choisir des options pour la création du controller. Ces options permettent d'avoir un controller avec des fonctions/méthodes par défaut.

Pour notre premier test, nous allons simplement mettre en place une méthode *Index* renvoyant simplement un string à afficher sur notre vue.

```
public class HomeController : Controller  
{  
  
    public string Index()  
    {  
        return "ASP.NET MVC3";  
    }  
  
}
```



Nous n'avons pas défini de vues pour l'instant mais nous avons cependant un résultat lorsque nous lançons notre application...

ASP.NET MVC 3 – PREMIÈRE APPLI

- Les Routes

MVC utilise le système de route de ASP.NET.

Ce système décide quel URL est mappé avec quel controller , action...

VS2010 a ajouté des routes par défaut.

Cela explique notre résultat précédent.

Ces routes sont définies dans le fichier *Global.asax*

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

    routes.MapRoute(
        "Default", // Nom d'itinéraire
        "{controller}/{action}/{id}", // URL avec des paramètres
        new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Paramètres par défaut
    );
}
```

ASP.NET MVC 3 – PREMIÈRE APPLI

Si nous désirons ajouter une nouvelle route, nous devrons simplement modifier notre méthode *RegisterRoutes* en ajoutant un mapping.

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

    routes.MapRoute(
        null,
        "Page{page}",
        new { Controller = "Product", action = "List" }
    );

    routes.MapRoute(
        "Default", // Nom d'itinéraire
        "{controller}/{action}/{id}", // URL avec des paramètres
        new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Paramètres par défaut
    );
}
```

ASP.NET MVC 3 – PREMIÈRE APPLI

- Vue

La sortie de notre exemple précédent est un simple string et non de l'HTML.

Nous allons donc modifier notre méthode index pour qu'elle nous renvoi un *vue*.

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

En faisant cela, nous informons notre modèle mvc qu'il doit nous afficher une vue.

En ne mettant pas de paramètre, ce sera la vue par défaut qui sera affichée

Erre du serveur dans l'application '/'.

La vue 'Index' ou son maître est introuvable, ou aucun moteur de vue ne prend en charge les emplacements où est effectuée la recherche. La recherche a porté sur les emplacements suivants :

~/Views/Home/Index.aspx
~/Views/Home/Index.ascx
~/Views/Shared/Index.aspx
~/Views/Shared/Index.ascx
~/Views/Home/Index.cshtml
~/Views/Home/Index.vbhtml
~/Views/Shared/Index.cshtml
~/Views/Shared/Index.vbhtml

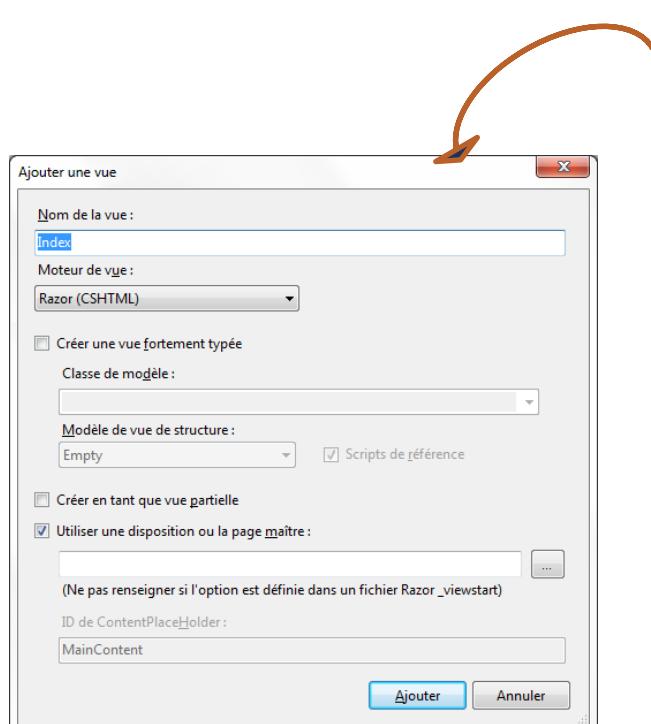
Ce message d'erreur apparait si aucune vue n'existe pour notre retour.

ASP.NET MVC 3 – PREMIÈRE APPLI

Ce message d'erreur nous informe donc qu'aucune vue n'est disponible MAIS également les endroits où MVC va rechercher.

En effet par convention, MVC va rechercher une vue portant le même nom que notre action dans le dossier *view*.

Nous allons donc ajouter notre vue



```
public ViewResult Index()
{
    return Vi
}
```

A yellow callout box highlights the 'Ajouter une vue...' button in the code editor's toolbar. Below it are two other buttons: 'Ajouter une vue...' and 'Aller à la vue'.

Nous avons la possibilité de lier notre vue à une *MasterPage* afin de profiter du layout.

Nous avons également la possibilité de créer une vue *fortement typée* dont nous reparlerons plus tard

ASP.NET MVC 3 – PREMIÈRE APPLI

Un fichier cshtml est créé pour nous.

- Spécifie au moteur *Razor* la layout à utiliser pour la mise en page
- cette partie est donc notre zone où nous pourrons ajouter du contenu à notre page HTML

```
Index.cshtml X Global.asax.cs
Objets et événements clients
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <title>Index</title>
</head>
<body>
    <div>
        </div>
    </body>
</html>
```

```
<html>
<head>
    <title>Index</title>
</head>
<body>
    <div>
        <h2>Bonjour, je suis votre vue</h2>
    </div>
</body>
</html>
```

Bonjour, je suis votre vue

ASP.NET MVC 3 – PREMIÈRE APPLI

Il y a d'autres possibilités comme retour d'action pour notre méthode *Index*

→ *RedirectResult*

Permet de rediriger vers une autre URL

→ *HttpUnauthorizedResult*

Permet de forcer l'authentification de l'utilisateur

Ces différents retour sont en fait des objets de type *ActionResult* qui permettent d'encapsuler et de réutiliser des actions commune.

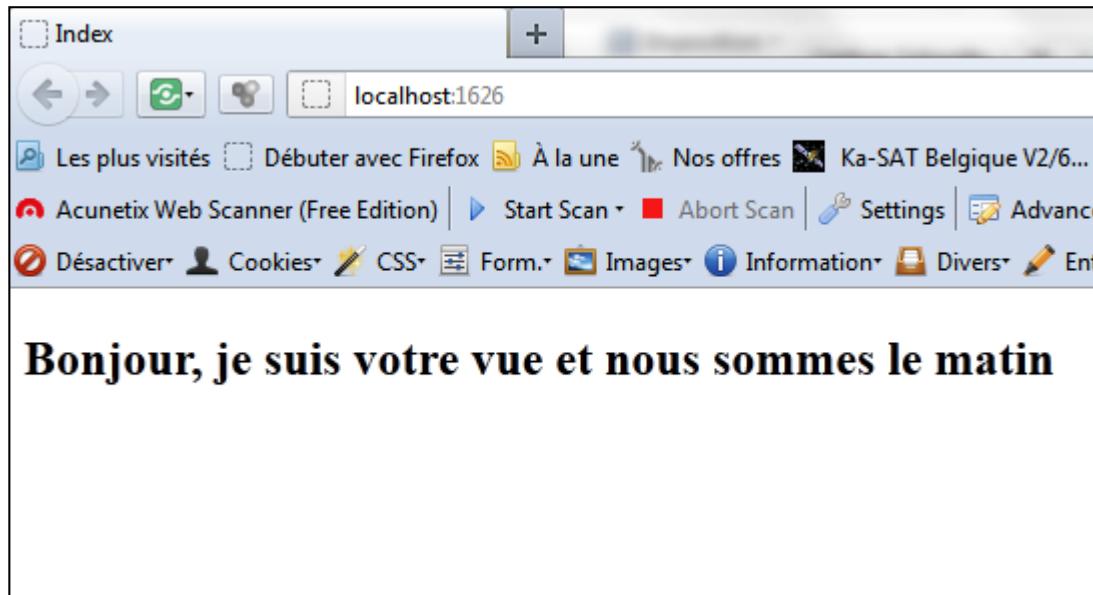
→ **Dynamisons notre sortie**

Il est bien sûr possible de produire le contenu de façon dynamique en modifiant notre méthode *Index* et notre vue

```
public ViewResult Index()
{
    int hour = DateTime.Now.Hour;
    ViewBag.Greeting = hour < 12 ? "le matin" : "l'après-midi";
    return View();
}
```

```
<div>
    <h2>Bonjour, je suis votre vue et nous sommes @ViewBag.Greeting</h2>
</div>
```

ASP.NET MVC 3 – PREMIÈRE APPLI



ASP.NET MVC 3 – DATA MODEL

La partie la plus importante dans le pattern MVC est la partie Model.

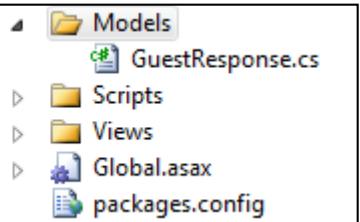
Le Domain Model, c'est la représentation des objets business (process, règles,).

Et ce sont nos vues et controllers qui exposent ce domaine.

Pour expliquer ces principes, nous allons mettre en place un formulaire permettant aux invités de confirmer leurs présence à une fête que nous organisons.

1. Notre modèle *GuestResponse*

```
public class GuestResponse
{
    public string Nom { get; set; }
    public string Email { get; set; }
    public string Telephone { get; set; }
    public bool? presence { get; set; }
}
```



1. Notre vue Home

```
<html>
<head>
    <title>Confirmation</title>
</head>
<body>
    <div>
        Veuillez suivre le lien ci-dessous afin de confirmer votre présence <br />
        @Html.ActionLink("Formulaire de confirmation", "ConfirmForm")
    </div>
</body>
</html>
```

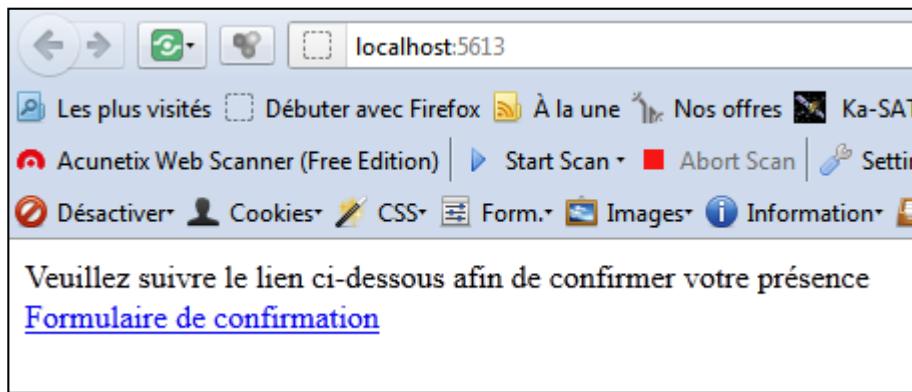
ASP.NET MVC 3 – DATA MODEL

`@Html.ActionLink` est une méthode *HTML Helper*.

Le framework MVC nous fournit une collection d'*Helper* nous permettant de produire les liens htmls, les text input, les checkbox,....

ActionLink prend deux paramètres :

- Le texte du lien
- L'action à réaliser après click



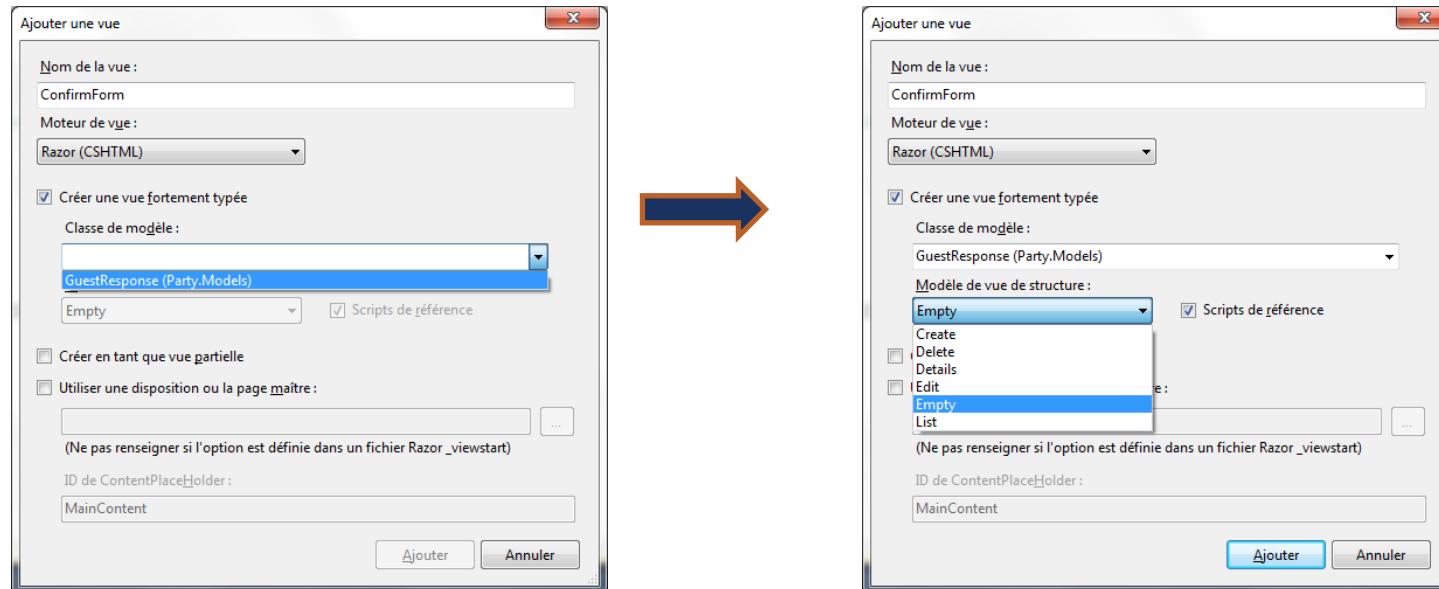
ASP.NET MVC 3 – DATA MODEL

Il nous reste à modifier notre controller pour pouvoir réagir à l'action de notre lien...

```
public ViewResult ConfirmForm()
{
    return View();
}
```

Et également à créer la vue correspondante.

Comme notre vue sera liée à un *Model*, nous devons créer une vue *fortement typée*.



ASP.NET MVC 3 – DATA MODEL

Nous complétons ensuite notre fichier *ConfirmForm.cshtml*

```
@model Party.Models.GuestResponse

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <title>ConfirmForm</title>
</head>
<body>
    <div>
        <div>
            @using (Html.BeginForm())
            {
                <p> Nom: @Html.TextBoxFor(x => x.Nom) </p>
                <p> Email: @Html.TextBoxFor(x => x.Email) </p>
                <p> Téléphone: @Html.TextBoxFor(x => x.Telephone) </p>
                <p> Serez-vous présent? <br/>
                    @Html.DropDownListFor(x => x.presence, new[] {
                        new SelectListItem() {Text = "Oui, je serais présent", Value = bool.TrueString},
                        new SelectListItem() {Text = "Non, je ne serait pas présent", Value = bool.FalseString}
                    }, "-") </p>
                <input type="submit" value="Valider ma participation" />
            }
        </div>
    </div>
</body>
</html>
```

ASP.NET MVC 3 – DATA MODEL

Html.BeginForm()

```
<form action="/Party/Home/ConfirmForm" method="post">
```

Html helper produisant un formulaire html dont l'action est configurée pour poster celui-ci vers la même url si aucun paramètre n'est passé.

Différentes signatures permettent de paramétriser le post

(extension) MvcForm HtmlHelper.BeginForm(**object routeValues**)

Écrit une balise <form> d'ouverture dans la réponse. Lorsque l'utilisateur enverra le formulaire, la demande sera traitée par une méthode d'action.

(extension) MvcForm HtmlHelper.BeginForm(**string actionPerformed**, string controllerName)

Écrit une balise <form> d'ouverture dans la réponse. Lorsque l'utilisateur enverra le formulaire, la demande sera traitée par une méthode d'action.

(extension) MvcForm HtmlHelper.BeginForm(**string actionPerformed**, string controllerName, FormMethod method)

Écrit une balise <form> d'ouverture dans la réponse. Lorsque l'utilisateur enverra le formulaire, la demande sera traitée par une méthode d'action.

Il existe 13 signatures différentes.

ASP.NET MVC 3 – DATA MODEL

Remarque

Contrairement à l'ASP.NET standard, le mécanisme de ViewState n'est pas implémenté. C'est pour cela que lorsqu'on poste le formulaire, nous perdons toutes les valeurs des champs au retour du server.

- **Traiter le formulaire**

Il nous faut pour cela ajouter :

- Une méthode répondant à une requête *http/GET*

Dans notre cas, elle servira à l'affichage initial du formulaire

- Une méthode répondant à une requête *http/POST*

Cette méthode sera responsable de la gestion des données postées par le formulaire

```
//Gestion du formulaire
[HttpGet]
public ViewResult ConfirmForm()
{
    return View();
}
[HttpPost]
public ViewResult ConfirmForm(GuestResponse guestResponse)
{
    return View("Merci", guestResponse);
}
```

ASP.NET MVC 3 – DATA MODEL

- Le principe de *Model Binding*

```
[HttpPost]
public ViewResult ConfirmForm(GuestResponse guestResponse)
{
    return View("Merci", guestResponse);
} ▲ 5 sur 8 ▼ ViewResult Controller.View(IView view, object model)
Crée un objet System.Web.Mvc.ViewResult qui restitue l'objet System.Web.Mvc.IView spécifié.
view: Vue restituée dans la réponse.
```

En utilisant cette signature, nous avons la possibilité de renvoyer via *HTTP/GET* le modèle qui aura été rempli.

La liaison entre le modèle et l'envoi de donnée via POST est appellé *Model Binding*

Le POST est parsé et transformé en une collection *Key/Value* qui est utilisée pour peupler le modèle lié.

C'est exactement l'inverse du *html helper* utilisé pour la création du formulaire

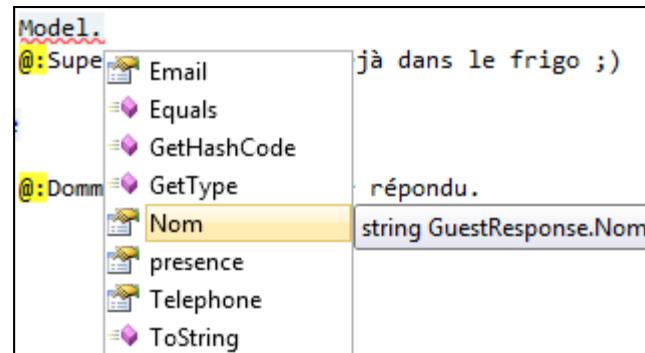
```
[HttpPost]
public ViewResult ConfirmForm(GuestResponse guestResponse)
{
    return View("Merci", guestResponse);
}
```

ASP.NET MVC 3 – DATA MODEL

Enfin, la vue *Merci* devra être créée de manière typée (pointant sur notre modèle) afin de pouvoir gérer les informations transmises

```
@model Party.Models.GuestResponse
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <title>Merci</title>
</head>
<body>
    <div>
        @if (Model.presence == true)
        {
            @:Super! La bière est déjà dans le frigo ;
        }
        else
        {
            @:Dommage! Merci d'avoir répondu.
        }
    </div>
</body>
</html>
```

Nous pouvons atteindre les propriétés du model via le mot clé *Model*



ASP.NET MVC 3 – DATA MODEL

- La validation

Dans une architecture MVC, la validation se place dans le *Domain Model* plutôt que dans l'interface utilisateur.

Cela permet de mettre en place la validation à un seul endroit et qu'elle soit appliquée pour tous les éléments utilisant ce modèle.

Pour mettre en place cette validation, nous devons importer l'espace de nom *System.ComponentModel.DataAnnotations*;

Et ensuite, nous pouvons utiliser les annotations sur nos propriétés

```
[Required(ErrorMessage = "Veuillez renseigner votre nom")]
public string Nom { get; set; }
[Required(ErrorMessage = "Veuillez renseigner votre mail")]
public string Email { get; set; }
[Required(ErrorMessage = "Veuillez renseigner votre Téléphone")]
public string Telephone { get; set; }
[Required(ErrorMessage = "Veuillez nous informer de votre présence")]
public bool? presence { get; set; }
```

Cela nous permettra de tester la validité de notre modèle via la méthode *ModelState.IsValid*

```
if (ModelState.IsValid)
{
    return View("Merci", guestResponse);
}
else
{
    return View();
}
```

ASP.NET MVC 3 – DATA MODEL

Du côté de notre *View*, nous avons la possibilité d'afficher facilement les messages d'erreur via L'*Html helper* → `@Html.ValidationSummary()`

```
@using (Html.BeginForm())
{
    <p>
        Nom: @Html.TextBoxFor(x => x.Nom)
    </p>
    <p>
        Email: @Html.TextBoxFor(x => x.Email)</p>
    <p>
        Téléphone: @Html.TextBoxFor(x => x.Telephone)</p>
    <p>
        Serez-vous présent?
        @Html.DropDownListFor(x => x.presence, new[] {
            new SelectListItem() {Text = "Oui, je serais présent", Value = bool.TrueString},
            new SelectListItem() {Text = "Non, je ne serait pas présent", Value = bool.FalseString}
        })
    </p>
    <input type="submit" value="Valider ma participation" />
    <br />
    <div style="border:1px solid #FF0000; width:300px; color:Red">
        @Html.ValidationSummary()
    </div>
}
```

Nom:

Email:

Téléphone:

Serez-vous présent?

- Veuillez renseigner votre nom
- Veuillez renseigner votre mail
- Veuillez renseigner votre Téléphone
- Veuillez nous informer de votre présence

ASP.NET MVC 3 – DATA MODEL

- Mettre en évidence les champs fautifs**

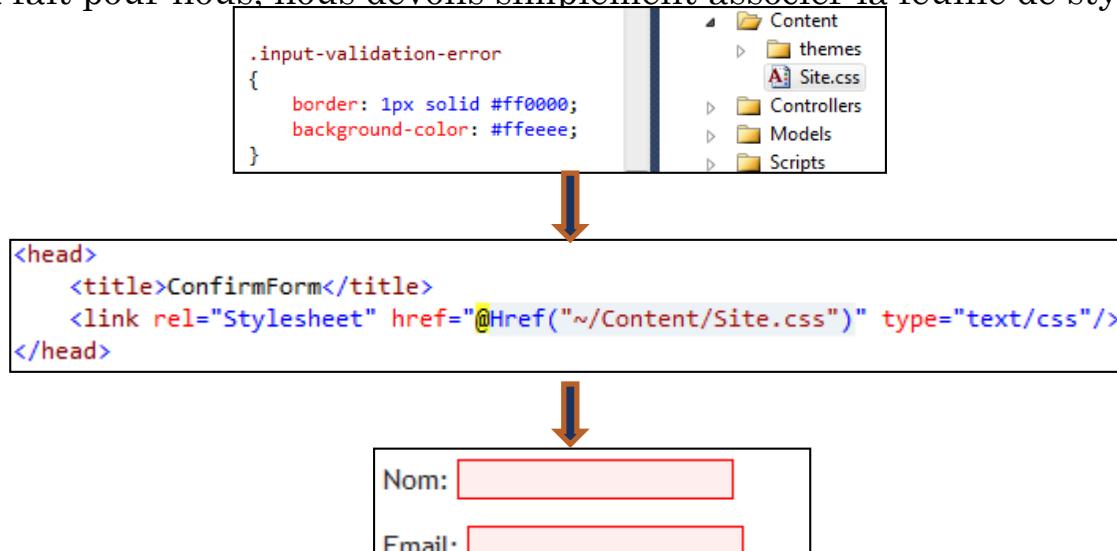
Le modèle mvc ne se contente pas de renvoyer la liste des erreurs vers notre vues.

Il change également l'html généré lorsqu'il détecte un champs fautif.

```
Nom: <input data-val="true" data-val-required="Veuillez renseigner votre nom" id="Nom" name="Nom">  
Email: <input class="input-validation-error" data-val="true" data-val-required="Veuillez renseig
```

Il nous suffit donc de créer un style css pour répercuter l'erreur au niveau visuel.

MVC l'a déjà fait pour nous, nous devons simplement associer la feuille de style à notre page



ASP.NET MVC 3 – DATA MODEL

- Un petit bonus : Envoi d'un mail

L'envoi d'un mail est simple grâce au helper WebMail...

```
@{  
    try  
    {  
        WebMail.SmtpServer = "smtp.example.com";  
        WebMail.SmtpPort = 587;  
        WebMail.EnableSsl = true;  
        WebMail.UserName = "mySmtpUsername";  
        WebMail.Password = "mySmtpPassword";  
        WebMail.From = "rsvps@example.com";  
        WebMail.Send("party-host@example.com", "Confirmation de votre présence",  
                    Model.Nom + " is " + ((Model.presence ?? false) ? "" : "not")  
                    + "attending");  
    }  
    catch (Exception)  
    {  
        @:<b>Impossible d'envoyer un mail.</b>  
    }  
}
```

En pratique, nous préférerons envoyer un mail à partir d'une action afin de ne pas mélanger notre code de rendu et notre process. (nous verrons cet exemple plus tard)



ASP.NET MVC 3 - RAZOR

Nous allons faire un rapide tour de la syntaxe Razor.

Le but de Razor est de minimiser le nombre de caractères requis à l'écriture d'un fichier de vues et d'introduire une nouvelle manière de coder qui est fluide et rapide. Au contraire de ce qu'on connaît déjà, le développeur n'a pas besoin de s'interrompre de coder pour dénoter explicitement les blocs serveur dans le HTML. Le parseur de Razor est assez intelligent afin d'inférer ceci à partir du code.

- L'extension des vues Razor est .cshtml pour les vues sous C# et .vbhtml pour les vues utilisant VB.NET.
- Les fichiers qui ne peuvent pas être affichés directement par la requête (les master pages, les vues partiels, etc.) sont précédés par un caractère “_” au début de leur noms. Donc si vous essayez d'afficher _Layout.cshtml qui est une Master Page vous obtiendrez une erreur serveur.

ASP.NET MVC 3 - RAZOR

Principalement la syntaxe de Razor est caractérisée par le caractère “@”. Nous allons parcourir quelques exemples de base pour découvrir sa syntaxe. Le but principal est de rendre les vues plus lisibles grâce aux enchainements du code et de HTML plus fluides.

Razor dispose des trois parseurs :

- Core parseur
- Parseur de Markup
- Parseur de Code.

Les trois parseurs travaillent ensemble pour analyser les vues de Razor.

- **Expressions**

L'expression la plus simple est d'afficher un message encodé :

```
@ViewBag.Message
```

Ou la version non encodée :

```
@Html.Raw(ViewBag.Message)
```

pour clairement indiquer à Razor qu'il doit traiter quelque on doit l'entourer avec des parenthèses :

```
<span>WGE@(widgetNumber)</span>
```

ASP.NET MVC 3 - RAZOR

- **Blocks de code**

Le code de block simple et multi ligne est contenu entre les accolades précédés par un caractère “@”:

```
@{ var @nom = "JASKULA";  
    var @prenom = "Thomas";}  
<p>Bonjour @prenom. Votre nome est @nom.</p>
```

- **Variables**

Pour déclarer les variables utilisez le mot clé “var” ou la syntaxe classique fortement typée. Le mot clé “new” est également autorisé.

```
var message = "Razor est coooooool";  
string title = "Syntaxe de Razor";  
int number = 1558;  
var stringBuilder = new System.Text.StringBuilder();
```

ASP.NET MVC 3 - RAZOR

• Chaines de caractères

Si dans votre chaîne des caractères il y a les backslash “\” ou les guillemets, vous pouvez utiliser l’opérateur “@” pour indiquer la chaîne verbatim.

```
var folder = @"C:\Files\ASP\MVC";
```

Pour les textes avec les guillemets, vous devez utiliser l’opérateur de chaîne verbatim est de doubler les guillemets comme dans l’exemple ci dessous :

```
var citation = @"""Thomas a dit : ""Utilisez Razor et vous serez libérés."""";
```

A l’intérieur du bloc de code, si vous voulez afficher le texte plein vous pouvez utiliser les caractères “@:” :

```
@{  
    @:Ceci est un texte plein.  
}
```

ASP.NET MVC 3 - RAZOR

• Commentaires

Dans Razor pour indiquer les commentaires vous commencez avec les caractères “@*” et terminez avec “*@”.

A l'intérieur de blocks de code vous avez la possibilité d'utiliser la syntaxe de commentaire de votre langage de programmation. Quelques exemples :

```
@{  
    @* Ceci est un commentaire *@  
    int age = 85;  
    @* Tout est commenté *@  
    @* int age = 85; *@  
    @* Ceci est le commentaire  
    multiligne  
    une autre ligne *@  
    /* commentaire multiligne  
    avec la syntaxe C# */  
}
```

ASP.NET MVC 3 - RAZOR

- **Instructions conditionnelles et les boucles**

- **If...then...else**

```
@{  
    var products = new string[] { "produit 1", " produit 2", "produit 3"  
    if (products.Count() > 0)  
    {  
        <p>Le premier produit est :@products[0]</p>  
    }  
    else{  
        <p>Il n'y a pas de produits en stock.</p> }  
    }  
}
```

ASP.NET MVC 3 - RAZOR

- **Switch**

```
@{  
var weekDay = 1;  
string message = "";  
switch (weekDay)  
{  
    case 1: message = "C'est lundi"; break;  
    case 2: message = "C'est mardi"; break;  
    default: message = "Jour inconnu."; break;  
}  
}
```

ASP.NET MVC 3 - RAZOR

- **For**

```
@{  
var products = new string[] { "product 1", "product 2", "product 3", "product 4" }; int cnt =  
products.Count();  
for (var i = 0; i < cnt; i++)  
{  
    <p>Le produit courant est : @products</p>  
}  
}
```

- **foreach**

```
@{  
var products = new string[] { "product 1", "product 2", "product 3", "product 4" };  
foreach (var product in products)  
{  
    <p>Le produit courant est : @product</p>  
}  
}
```

URLS, ROUTING & AREAS

151

URLS, ROUTING & AREAS

Pour capturer les urls MVC, ASP.NET utilise le système de routing.

Le système de routing a deux fonctions :

1. Examiner les requêtes entrantes et définir quel Controller et quelle action est demandée
2. Générer les Urls de sortie (pour les liens par exemple)

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    RegisterGlobalFilters(GlobalFilters.Filters);
    RegisterRoutes(RouteTable.Routes);
}

public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        "Default", // Nom d'itinéraire
        "{controller}/{action}/{id}", // URL avec des paramètres
        new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Paramètres par défaut
    );
}
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());
}
```

URLS, ROUTING & AREAS

- Url Patterns

Nous n'avons pas à écrire toutes les combinaisons possibles pour les urls.

En effet, chaque route contient un Url pattern qui est comparé à l'url demandée.

Si le pattern correspond, il est utilisé pour résoudre l'url.

Les urls peuvent être décomposées en segments.

Chaque segments est séparé par un /.

Le système de routing analyse chaque segment



URLS, ROUTING & AREAS

Request	Segments
http://mysite.com/Admin/Index	controller = Admin action = Index
http://mysite.com/Index/Admin	controller = Index action = Admin
http://mysite.com/Apples/Oranges	controller = Apples action = Oranges
http://mysite.com/Admin	Erreur - trop peu de segment
http://mysite.com/Admin/Index/Soccer	Erreur - trop de segment

Par défaut :

- Mvc n'utilise le pattern que pour les urls qui correspondent exactement
- Si il y a le bon nombre de segment, il utilise de manière littérale ceux-ci pour trouver le controller et l'action

URLS, ROUTING & AREAS

- Définir des valeurs par défaut

La valeur par défaut est utilisé lorsque l'url ne match avec aucun pattern

```
routes.MapRoute(  
    "Default", // Nom d'itinéraire  
    "{controller}/{action}/{id}", // URL avec des paramètres  
    new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Paramètres par défaut  
)
```

- Segment static

Tous les segments ne sont pas nécessairement variables.

Si nous désirons supporter l'url suivante

<http://mydomain.com/Public/Home/Index>

Nous écrirons

```
routes.MapRoute("", "Public/{controller}/{action}",new { controller = "Home", action = "Index" });
```

Remarque :

Les routes sont appliquées dans l'ordre de leurs déclarations dans le fichier *Global.asax*

URLS, ROUTING & AREAS

- **Utiliser les zones**

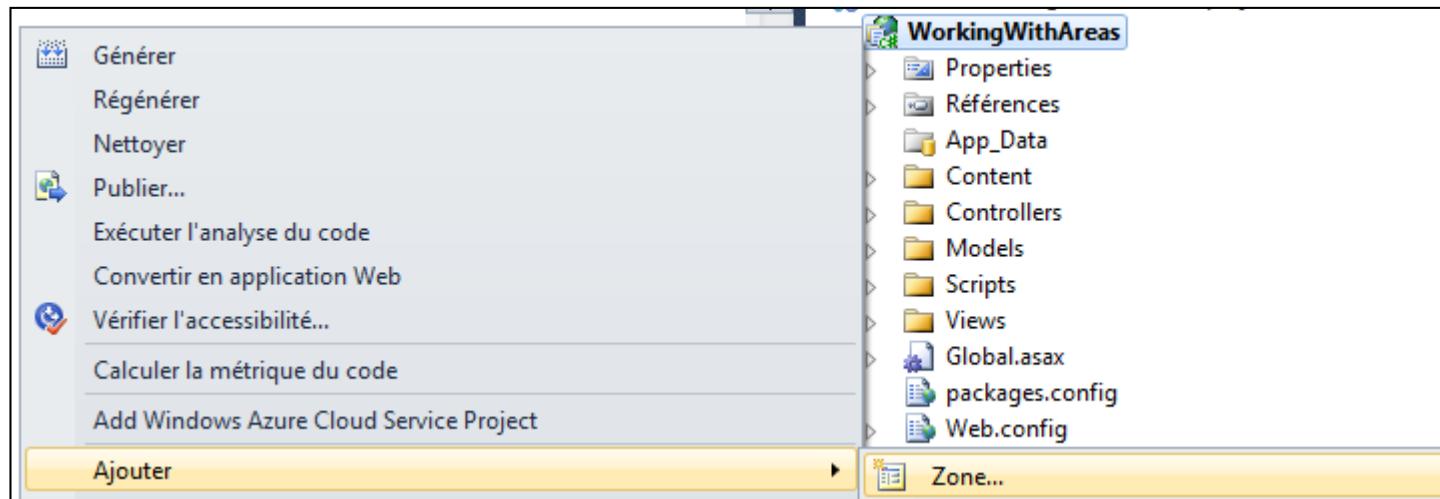
MVC supporte l'organisation des applications webs en *Area*.

Chaque *Area* représente un segment fonctionnel comme l'administration, le paiement, les clients, les supports,....

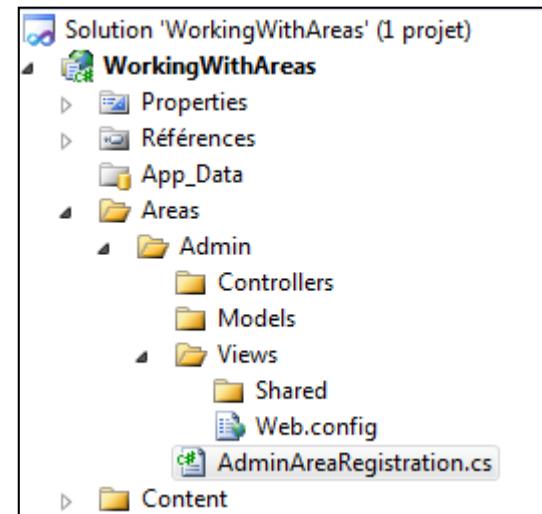
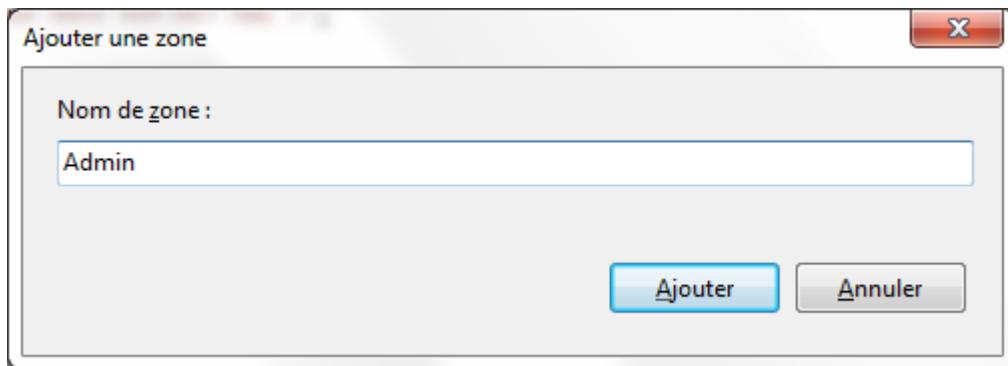
Ces *Area* facilitent la maintenance des larges applications.

Ces *Areas* sont supportés par le système de routing.

La création des *Areas* est simple



URLS, ROUTING & AREAS



Dans la zone, nous pouvons remarquer les dossiers d'un mini-mvc.

Controller et Model sont vide mais le dossier view contient un autre dossier *Shared* et un fichier web.config qui permet de configurer le moteur de vue.

Le dossier shared contiendra des vues partiels sur lesquels nous reviendrons.

URLS, ROUTING & AREAS

Nous avons également *AdminAreaRegistration.cs*.

```
namespace WorkingWithAreas.Areas.Admin
{
    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 }
            );
        }
    }
}
```

Dans la méthode *RegisterArea*, nous avons une route définie. Cette zone nous permettra de définir des routes applicable uniquement dans notre zone

URLS, ROUTING & AREAS

Cette méthode sera automatiquement appelée par *Application_start* se trouvant dans *Global.asax*.

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();

    RegisterGlobalFilters(GlobalFilters.Filters);
    RegisterRoutes(RouteTable.Routes);
}
```

Remarque : NE MODIFIER PAS l'ordre car les zones doivent être connues avant l'enregistrement des routes

URLS, ROUTING & AREAS

Remplissage des zones

Il faut considérer les zones comme étant des sous-dites.

Nous appliquerons donc la technique habituelle pour la création de notre contenu dans celles-ci.

Erreur du serveur dans l'application '/'.

Plusieurs types correspondant au contrôleur nommé 'Home' ont été trouvés. Cela peut se produire si la route qui traite cette demande ('{controller}/{action}/{id}') ne spécifie pas d'espaces de noms pour rechercher un contrôleur correspondant à la demande. Si tel est le cas, enregistrez cette route en appelant une surcharge de la méthode 'MapRoute' qui prend un paramètre 'namespaces'.

*La demande pour 'Home' a trouvé les contrôleurs correspondants suivants :
WorkingWithAreas.Controllers.HomeController
WorkingWithAreas.Areas.Admin.Controllers.HomeController*

URLS, ROUTING & AREAS

Quand une route est définie , celle-ci est définie pour son *namespace*.

Malheureusement, l'enregistrement dans le *Global.asax* n'est pas limitée par ce concept.

Il nous faut donc définir une priorité afin que le système puisse trouver le controller adéquat et ne nous envoi pas l'erreur d'ambiguïté.

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

    routes.MapRoute(
        "Default", // Nom d'itinéraire
        "{controller}/{action}/{id}", // URL avec des paramètres
        new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Paramètres par défaut
        ,new[] { "WorkingWithAreas.Controllers" }

    ▲ 5 sur 6 ▼ (extension) Route RouteCollection.MapRoute(string name, string url, object defaults, string[] namespaces)
        Mappe l'itinéraire d'URL spécifié et définit des valeurs d'itinéraire par défaut et des espaces de noms.
        namespaces: Ensemble d'espaces de noms de l'application.
}
```

Ce simple changement permet au système de choisir le Controller se trouvant dans le namespace principal en priorité si un conflit survient.

URLS, ROUTING & AREAS

Générer les liens dans les zones

Il n'y a rien de particulier à faire!

Le framework MVC détecte automatiquement la zone dans laquelle le lien est généré.

```
<div>
    <h2>Admin Area index</h2>
    <br />
    @Html.ActionLink("Click me", "About")
</div>
```



Admin Area index

[Click me](#)

http://localhost:8619/WorkingWithAreas/Admin/Home/About

`@Html.ActionLink("vers la zone cliente", "Index", new { area = "Client" })`

Pour changer de zones, nous passerons par une variable appelée *area*.

Ou les *routed link*

```
@Html.RouteLink("Partie Cliente", "Client_default", new { action = "Index", Controller = "Home" })
```

▲ 4 sur 11 ▼ (extension) MvcHtmlString HtmlHelper.RouteLink(string linkText, object routeValues, object htmlAttributes)

Retourne un élément d'ancre (élément a) qui contient le chemin d'accès virtuel de l'action spécifiée.

htmlAttributes: Objet qui contient les attributs HTML à définir pour l'élément.

URLS, ROUTING & AREAS

Les bonnes pratiques pour nos URLs

1. Urls propres et lisibles

Les utilisateurs aime pouvoir atteindre facilement une url.

Essayez donc d'obtenir les urls les plus simple et lisible possible.

~~http://www.monsite.com/Pro-ASP-NET-MVC-3-Framework/dp/1430234040/ref=sr_1_13?s=books&ie=UTF8&qid=1294771153&sr=1-13~~



<http://www.monsite.com/books/pro-aspnet-MVC-framework>

URLS, ROUTING & AREAS

➤ Quelques Guidelines

- Ecrivez les urls pour qu'elles décrivent le contenu et pas l'implémentation des pages
- Préférez des titres plutôt que des id (article/2563 → /article/VeloFemme)
- Créez une hiérarchie logique pour le visiteur (articles/sport/femme/velo)
- Ne mettez pas de majuscules
- Evitez les symboles dans l'url (-, #, espace, +,...)
- Soyez consistent

Vous trouverez quelques conseils sur <http://www.w3.org/Providers/Style/URI>

CONTROLLERS & ACTIONS

165

CONTROLLERS & ACTIONS

Un controller sous le framework MVC doit implémenter l'interface *IController*.

```
namespace System.Web.Mvc
{
    // Résumé :
    //     Définit les méthodes requises pour un contrôleur.
    public interface IController
    {
        // Résumé :
        //     Exécute le contexte de la requête spécifiée.
        //
        // Paramètres :
        //     requestContext;
        //     Contexte de la requête.
        void Execute(RequestContext requestContext);
    }
}
```

Il est cependant recommandé (pour des raisons de facilité) de faire simplement hériter notre controller de la classe *Controller*

CONTROLLERS & ACTIONS

La classe controller fournit 3 fonctionnalités

- *Méthodes d'Action*
- *Les résultats d'Action*
- *Les Filtres*

Lorsque nous créons un Controller, nous profitons de cette classe qui implémente la méthode *Execute* et qui a la responsabilité d'appeler la bonne action suivant notre route.

La classe controller fait également le lien avec la vue de façons logique :

- 1) HomeController.css → Home/*lesvues*
- 2) Action Index → Home/Index.cshtml

CONTROLLERS & ACTIONS

- La réception des données

Un controller a besoin d'accéder aux entrée de données (chaine de caractères, formulaires, paramètres url,...)

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

Si on créé un controller 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 context d'exécution.

CONTROLLERS & ACTIONS

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

CONTROLLERS & ACTIONS

Propriétés	Type	Description
HttpContext.Item	IDictionnary	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 context
    string userName = User.Identity.Name;
    string serverName = Server.MachineName;
    string clientIP = Request.UserHostAddress;
    DateTime dateStamp = HttpContext.Timestamp;
    string oldProductName = Request.Form["Nom"];
    string newProductName = Request.Form["Prenom"];
    return View("ProductRenamed");
}
```

CONTROLLERS & ACTIONS

2) Utiliser les paramètres de la méthode Action



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

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

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

Mais comment fait-il ?

CONTROLLERS & ACTIONS

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 »,...)

CONTROLLERS & ACTIONS

- 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*.

Voici un exemple d'utilisation de *ExecuteResult* (ActionAndResult.sln)

```
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);
    }
}
```

CONTROLLERS & ACTIONS

```
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");
    }
}
```

CONTROLLERS & ACTIONS

Le framework MVC nous propose des types *tout-fait*

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

CONTROLLERS & ACTIONS

Type	Controller helper
HttpNotFoundResult	HttpNotFound
HttpStatusCodeResult	None
EmptyResult	None

- Rendu HTML au travers d'une vue**

La plus commune des réponses.

```
public ViewResult Index()
{
    return View("HomePage");
}
▲ 4 sur 8 ▼ ViewResult Controller.View(string viewName)
Crée un objet System.Web.Mvc.ViewResult à l'aide du nom de vue qui restitue une vue.
viewName: Nom de la vue restituée dans la réponse.
```

```
public ViewResult Index()
{
    return View("~/Views/Other/Index.cshtml");
}
```

CONTROLLERS & ACTIONS

- Fournir un model à la vue

```
public ViewResult Index()
{
    DateTime date = DateTime.Now;
    return View(date);
} ▲ 3 sur 8 ▼ ViewResult Controller.View(object model)
Crée un objet System.Web.Mvc.ViewResult à l'aide du modèle qui restitue une vue dans la réponse.
model: Modèle qui est restitué par la vue.
```

```
<body>
    <div>
        The day is: @((DateTime)Model).DayOfWeek
    </div>
</body>
```

Si il s'agit d'une classe fortement typé

```
@model DateTime
@{
    Layout = null;
    ViewBag.Title = "Index";
}
<html>
<body>
    <div>
        The day is: @Model.DayOfWeek
    </div>
</body>
</html>
```

CONTROLLERS & ACTIONS

- Passer des valeur grâce au 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 ViewResult Index()
{
    ViewBag.Message = "Hello";
    ViewBag.Date = DateTime.Now;
    return View();
}
```



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

Passer des valeur grâce au ViewData

C'est l'ancienne version qui existait pour passer les données avant MVC. Il s'agit d'un simple dictionnaire.

```
public ViewResult Index()
{
    ViewData["Message"] = "Hello";
    ViewData["Date"] = DateTime.Now;
    return View();
}
```



```
<div>
    <h2>Admin Area index</h2>
    <br />
    @ViewData["Date"] , @ViewData["Message"]
</div>
```

CONTROLLERS & ACTIONS

- Effectuer une redirection**

Lorsqu'on effectue une redirection on envoi une en-tête http :

- 301 : redirection temporaire
- 302 : redirection permanente

Simple

```
public RedirectResult Redirect()
{
    return Redirect("/Example/Index");
}
```

Vers une action du Controller courant

```
public RedirectToRouteResult Redirect()
{
    return RedirectToAction("Index");
}
```

permanente

```
public RedirectToRouteResult Redirect()
{
    return RedirectToAction("Index", "MyController");
}
```

```
public RedirectToRouteResult Redirect()
{
    return RedirectToRoute(new
    {
        controller = "Example",
        action = "Index",
        ID = "MyID"
    });
}
```

De manière

```
public RedirectToRouteResult Redirect()
{
    return RedirectToActionPermanent("Index", "MyController");
}
```

CONTROLLERS & ACTIONS

Astuce

Lors d'une redirection, c'est une nouvelle requête http qui est envoyée vers le browser.

Nous perdons donc les informations de la requête originale.

Pour passer les données à la redirection, on peut utiliser *TempData*.

TempData est similaire aux session excepté que *TempData* est marqué pour suppression quand les données seront lues.

```
public RedirectToRouteResult Redirect()
{
    TempData["Message"] = "Hello";
    TempData["Date"] = DateTime.Now;
    return RedirectToAction("Index");
}
```



```
public ViewResult Index()
{
    ViewBag.Message = TempData["Message"];
    ViewBag.Date = TempData["Date"];
    return View();
}
```

Si on désire relire une seconde fois les valeurs du *TempData*, il nous faudra utiliser la méthode *Keep*. Cette méthode permet de re-lire la donnée avant qu'elle soit à nouveau marquée pour suppression.

(TempData.Keep(); TempData.Keep(▲ 2 sur 2 ▼ void TempDataDictionary.Keep(string key) Marque la clé spécifiée dans le dictionnaire pour la rétention. key: Clé à conserver dans le dictionnaire.
--

Nous pouvons également utiliser la méthode *Peek* pour empêcher la suppression

TempData.Peek() object TempDataDictionary.Peek(string key) Retourne un objet qui contient l'élément associé à la clé spécifiée, sans marquer la clé pour la suppression. key: Clé de l'élément à retourner.
--

CONTROLLERS & ACTIONS

- Renvoyer du texte

En plus du HTML, il est possible de renvoyer d'autres contenus :

- XML
- JSON
- CSV
- Juste du texte

Excepté pour Json (nous y reviendrons plus tard), nous utiliserons *ContentResult*

```
return Content(  
//return
```

▲ 3 sur 3 ▼ ContentResult Controller.Content(string content, string contentType, System.Text.Encoding contentEncoding)
Crée un objet résultat de contenu à l'aide d'une chaîne, du type de contenu et de l'encodage du contenu.
content: Contenu à écrire dans la réponse.

```
string message = "Texte normal";  
return Content(message, "text/plain", Encoding.Default);
```

```
StoryLink[] stories = GetAllStories();  
XElement data = new XElement("StoryList", stories.Select(e =>  
{  
    return new XElement("Story",  
        new XAttribute("title", e.Title),  
        new XAttribute("description", e.Description),  
        new XAttribute("link", e.Url));  
});  
return Content(data.ToString(), "text/xml");
```

CONTROLLERS & ACTIONS

En ce qui concerne le JSON, le framework MVC possède une classe JsonResult qui permet de sérialiser les objets .NET en format Json correct.

```
[HttpPost]
public JsonResult JsonData()
{
    StoryLink[] stories = GetAllStories();
    return Json(stories);
}
```

Remarque :

Pour des raisons de sécurité, *JsonResult* n'autorise que le POST de la génération.

CONTROLLERS & ACTIONS

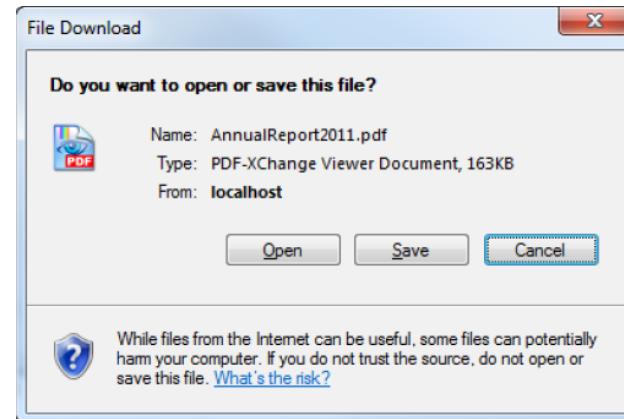
- Renvoyer un fichier ou des données binaires**

Pour cela nous utiliserons la classe abstraite *FileResult*.

3 classes concrète existe :

- FileContentResult* qui envoi directement un fichier à partir du file system
- FileStreamResult* qui envoi le contenu d'un tableau de bytes
- FileResult* qui envoi le contenu d'un *System.IO.Stream* qui est déjà ouvert

```
public FileResult AnnualReport()
{
    string filename = @"c:\AnnualReport.pdf";
    string contentType = "application/pdf";
    string downloadName = "AnnualReport2011.pdf";
    return File(filename, contentType, downloadName);
}
```



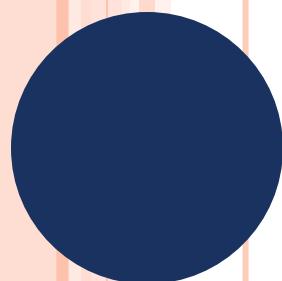
```
public FileContentResult DownloadReport()
{
    byte[] data = ...
    return File(data, "application/pdf", "AnnualReport.pdf");
}
```

```
public FileStreamResult DownloadReport()
{
    Stream stream = ...
    return File(stream, "text/html");
}
```

CONTROLLERS & ACTIONS

- Customizer un ActionResult

Cfr **CustomActionResult.sln**



185

VUES

VUES

- **Html Helpers**

Il est intéressant de pouvoir créer ses propres *html helper* permettant de ne pas devoir répéter les même tags razor et html...

Nous pouvons les comparer aux *Users controls*.

Nous avons déjà vu un *html helper* nous permettant de construire un formulaire à partir d'un modèle, un autre permettre la validation et un dernier pour créer un lien.

(@Html.BeginForm(), @Html.ValidationSummary(), @Html.ActionLink)

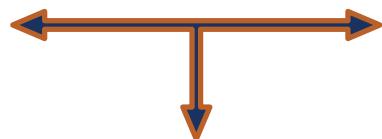
Il existe deux type d'html helpers :

VUES

- Création d'un *Inline Html Helper*

La façon la plus rapide de créer un helper est d'utiliser le tag @helper directement dans une vue.

```
@helper CreateList(string[] items)
{
    <ul>
        @foreach (string item in items)
        {
            <li>@item</li>
        }
    </ul>
}
<h4>InlineHelper</h4>
Days of the week: <p/>
@CreateList(ViewBag.Days)
<p />
Fruit I like: <p />
@CreateList(ViewBag.Fruits)
```



InlineHelper

Days of the week:

- Lundi
- Mardi
- Mercredi
- Jeudi
- Vendredi
- Samedi
- Dimanche

Fruit I like:

- Pommes
- Poires
- Abricots

```
public ActionResult Index()
{
    string[] days = { "Lundi", "Mardi", "Mercredi",
                      "Jeudi", "Vendredi", "Samedi",
                      "Dimanche" };
    string[] Fruits = { "Pommes", "Poires", "Abricots" };
    ViewBag.Days = days;
    ViewBag.Fruits = Fruits;

    return View();
}
```

VUES

- **Création d'un *External Html Helper***

Le problème du type d'helper précédent est qu'il ne peut être utilisé que dans la vue où il est déclaré.

De plus, si le code augmente en complexité, cela peut devenir difficilement gérable.

L'alternative est donc de créer un helper externe en C#.

```
public static class CustomHtmlHelpers
{
    public static MvcHtmlString List(this HtmlHelper html, string[] listItems)
    {
        TagBuilder tag = new TagBuilder("ul");
        foreach (string item in listItems)
        {
            TagBuilder itemTag = new TagBuilder("li");
            itemTag.SetInnerText(item);
            tag.InnerHtml += itemTag.ToString();
        }
        return new MvcHtmlString(tag.ToString());
    }
}
```

VUES

Les paramètres qui suivent permettent de passer des valeurs de la vue vers l'*Helper*.

La façon la plus simple de créer de l'*HTML*, sans devoir s'inquiéter des caractères spéciaux et ceux d'échappement, est d'utiliser la classe *TagBuilder*.

```
TagBuilder tag = new TagBuilder("ul");
```

Membre	Description
InnerHtml	Permet de renvoyer directement de l'HTML contenu dans un string
SetInnerText(string)	Permet de spécifier le contenu texte d'un élément html. Le string en paramètre est automatiquement encodé pour permettre un affichage sur.
AddCssClass	Permet d'ajouter une classe css à un élément html
MergeAttribute(string, string, bool)	Ajout d'un attribut à un élément html. Le premier paramètre est le nom de l'attribut, le deuxième sa valeur, le dernier si l'attribut existant doit être remplacé

VUES

- **Les Html Helpers natifs**

Outre les Helpers `@Html.BeginForm()`, `@Html.ValidationSummary()`, `@Html.ActionLink` que nous avons déjà utilisés.

Il existe également :

- **WebGrid Helper**

Il nous permet d'afficher des données sous forme de grille en utilisant l'élément table de l'HTML.

Il existe beaucoup de configurations possibles et il est donc plus difficile de décrire cet *Helper* que de l'utiliser.

Nous allons donc voir par un exemple son utilisation.

VUES

1. Création de la séquence d'objets à afficher

```
public ActionResult Grid()
{
    IEnumerable<Product> productList = new List<Product> {
        new Product {Name = "Kayak", Category = "Watersports", Price = 275m},
        new Product {Name = "Lifejacket", Category = "Watersports", Price = 48.95m},
        new Product {Name = "Soccer ball", Category = "Football", Price = 19.50m},
        new Product {Name = "Corner flags", Category = "Football", Price = 34.95m},
        new Product {Name = "Stadium", Category = "Football", Price = 79500m},
        new Product {Name = "Thinking cap", Category = "Chess", Price = 16m}
    };
    return View(productList);
}
```

1. Création de la séquence d'objets à afficher

```
@model IEnumerable<WebGridHelper.Models.Product>
 @{
    var grid = new WebGrid(
        source: Model,
        rowsPerPage: 4);
}
@grid.GetHtml(
    tableStyle: "grid",
    headerStyle: "header",
    rowStyle: "row",
    footerStyle: "footer",
    alternatingRowStyle: "altRow",
    columns: grid.Columns(
        grid.Column("Name", "Item", style: "textCol"),
        grid.Column("Price", style: "numberCol",
            format: @<text>$@string.Format("{0:F2}", item.Price) </text>)
    )
)
```

VUES

Notre vue est composée de deux blocs.

Le premier bloc définit l'objet grid en lui passant notre modèle et le nombre d'enregistrement à afficher par page.

Nous avons donc une pagination de façon toute simple.

<u>Item</u>	<u>Price</u>
Kayak	\$275,00
Lifejacket	\$48,95
Soccer ball	\$19,50
Corner flags	\$34,95
1	2
	3

Paramètre	Valeur par défaut	Description
Source	null	Renseigne la source contenant les données à afficher
ColumnNames	null	Définit les paramètres de la source de données qui serviront pour les colonnes
rowsPerPage	10	Définit le nombre de lignes qui seront affichées
canPage	true	Active ou désactive la pagination
canSort	true	Active ou Désactive la possibilité de trier les colonnes en cliquant sur les en-têtes

VUES

Le second bloc définit la configuration visuelle du grid en spécifiant notamment :

- les styles visuels à utiliser pour le header, les lignes,...
- L'intitulé des colonnes
- Le texte des lignes généré à partir du model

Paramètre	Valeur par défaut	Description
tableStyle	null	Css pour l'élément table
headerStyle	null	Css pour le header <tr>
footerStyle	null	Css pour le footer <tr>
rowStyle	null	Css pour les lignes
alternatingRowStyle	null	Css alternative pour les lignes
caption	null	Texte affiché avant la table
displayHeader	null	Active ou désactive les en-têtes
fillEmptyRows	False	Si à vrai , des lignes vides sont ajoutées en fin de table

VUES

Paramètre	Valeur par défaut	Description
previousText	null	Texte pour le bouton précédent
nextText	null	Texte pour le bouton suivant
numericLinksCount	null	Spécifie le nombre de page qui doivent être affichées dans le footer
columns	null	Permet de configurer individuellement les colonnes

Tips :

```
ViewBag.WebGrid = new WebGrid(source: productList, rowsPerPage: 4);
```

Nous pouvons déclarer directement notre grid dans notre C#.

Et l'utiliser dans la vue comme suit

COGNITIC

VUES

➤ Chart Helper

Cet helper permet de produire des graphs.

Pour toutes les options, les styles, etc.. je vous conseille de consulter MSDN.

➤ Création d'une image(ChartHelper.sln - ChartImage)

La première étape est de créer l'objet chart

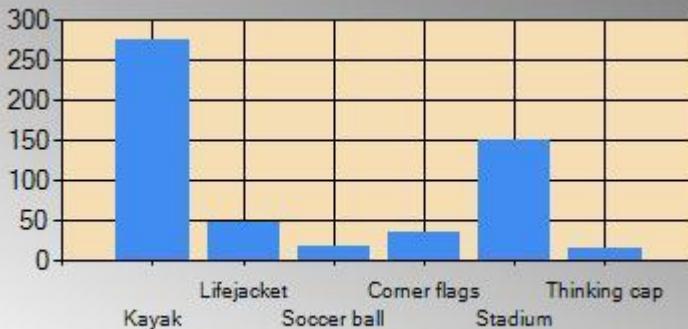
```
Chart chart = new Chart(400, 200,
    @"<Chart BackColor=""Gray"" BackSecondaryColor=""WhiteSmoke"""
    BackGradientStyle=""DiagonalRight"" AntiAliasing=""All"""
    BorderlineDashStyle = ""Solid"" BorderlineColor = ""Gray"""
    <BorderSkin SkinStyle = ""Emboss"" />
    <ChartAreas>
        <ChartArea Name=""Default"" _Template_=""All"" BackColor=""Wheat"""
        BackSecondaryColor=""White"" BorderColor=""64, 64, 64, 64"""
        BorderDashStyle=""Solid"" ShadowColor=""Transparent""/>
    </ChartArea>
    </ChartAreas>
)</Chart>");
```

COGNITIC La configuration visuelle est basée sur un XML. Il y a 35 styles différents pour générer les graphs en 2D,3D et configurer l'apparence de chaque élément. 195

VUES

La seconde étape consiste à ajouter les séries pour peupler le graph.

```
chart.AddSeries(  
    chartType: "Column",  
    yValues: productList.Select(e => e.Price).ToArray(),  
    xValue: productList.Select(e => e.Name).ToArray());
```



Tips:

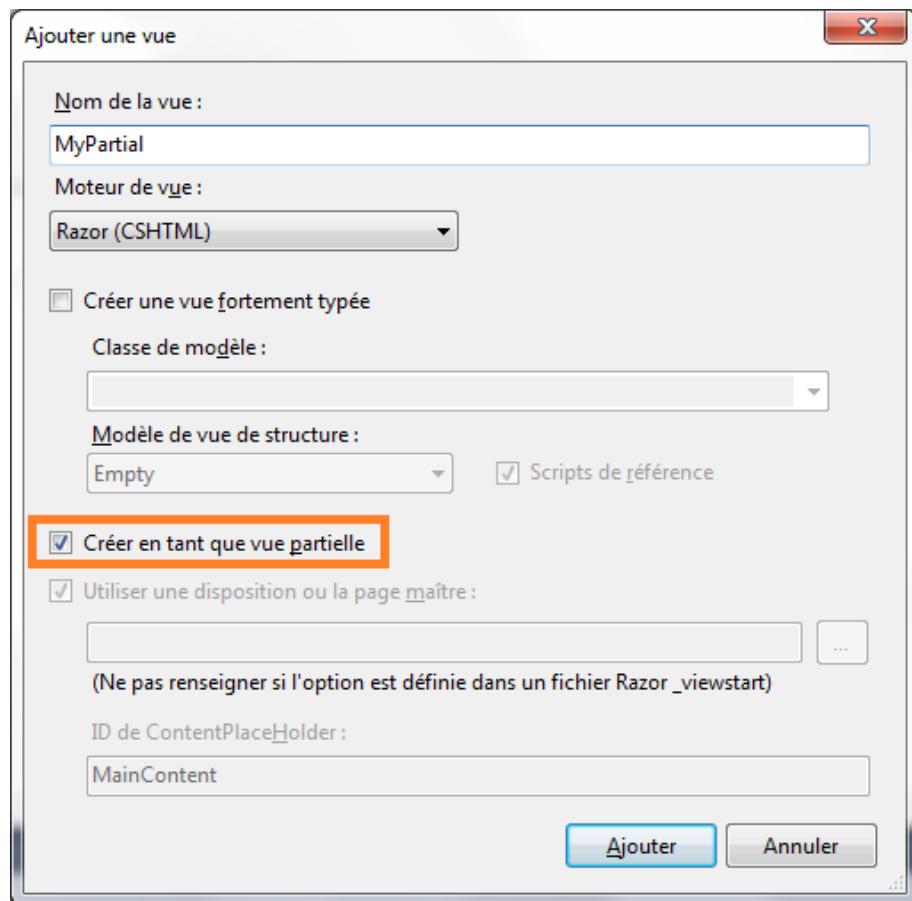
Il est recommandé d'utiliser une partial vue pour les graphs

LES VUES PARTIELLES

Il est fréquent d'utiliser un même fragment de code @Razor dans plusieurs pages.

Plutôt que de *copier-coller*,
nous pouvons utiliser les vues
partielles.

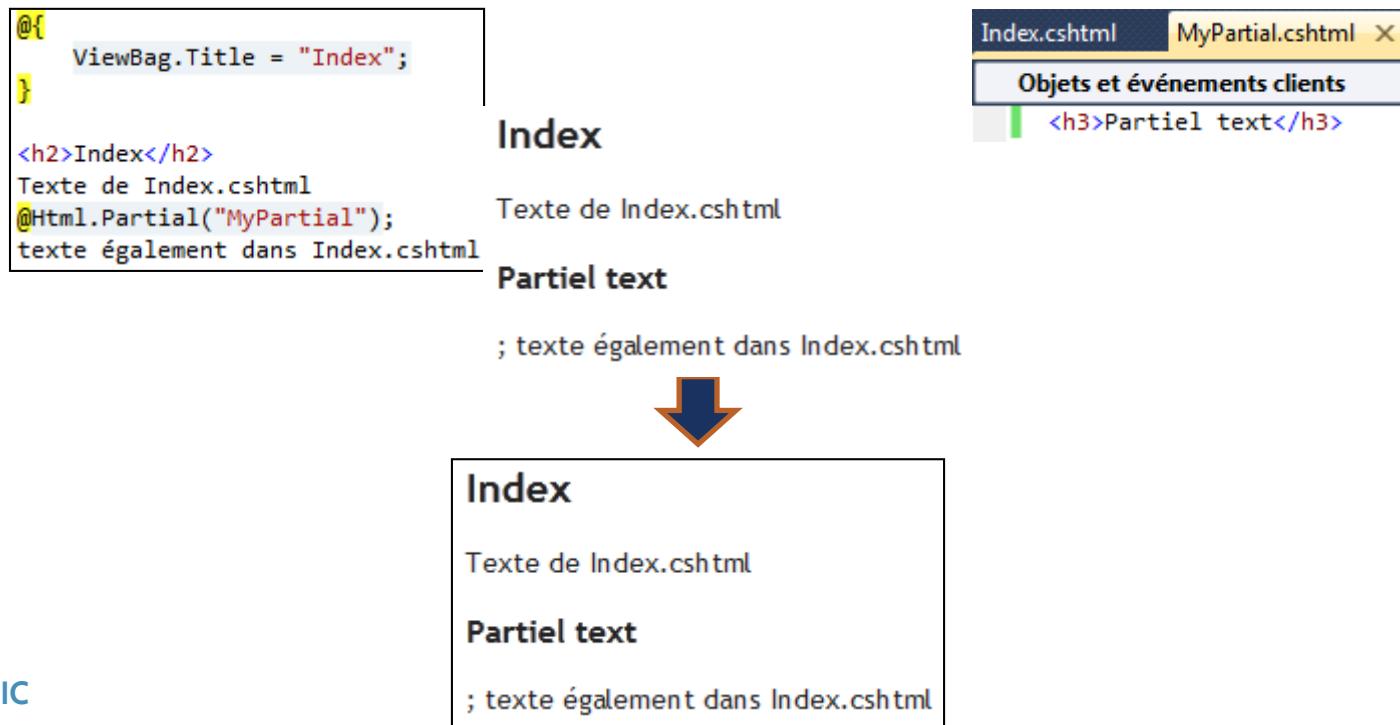
La création d'une vue partielle
n'est pas très compliquée :



LES VUES PARTIELLES

Une vue partielle est vide lorsqu'elle est créée.

Pour l'inclure dans notre vue principale, il suffit d'utiliser *html.Partial()*



LES VUES PARTIELLES

- Utiliser les Child Action**

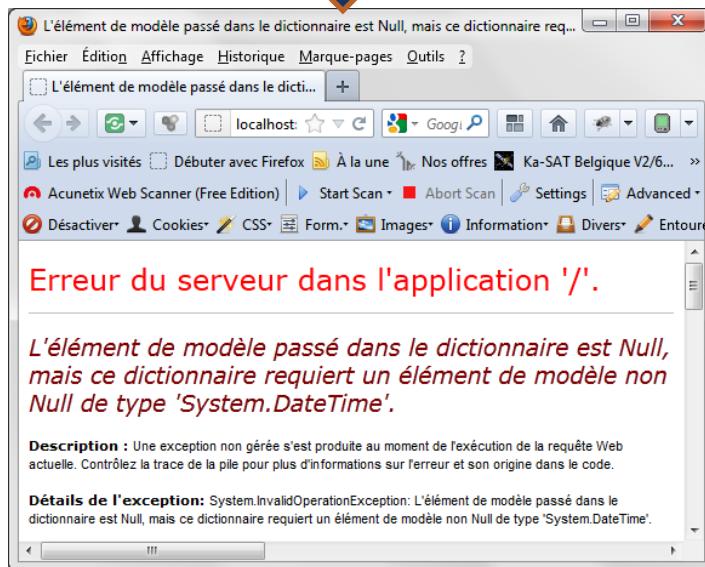
Les *Child Action* sont utilisées pour permettre des actions pour les vues partielles.

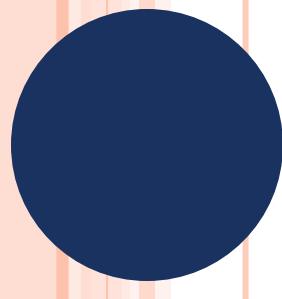
Pour les créer :

```
[ChildActionOnly]
public ActionResult Time()
{
    return PartialView(DateTime.Now);
}
```



Index.cshtml	MyPartial.cshtml	Global.asax.cs	Home
Objets et événements clients			
@model DateTime			
<p>The time is: @Model.ToShortTimeString()</p>			





200

SECTIONS



LES SECTIONS

FILTRES

202

FILTRES

Les filtres permettent de réduire la redondance de code...

Prenons un exemple : (introFiltres.sln)

```
public ActionResult Index()
{
    if (!Request.IsAuthenticated)
    {
        FormsAuthentication.RedirectToLoginPage();
    }
    return View();
}

public ActionResult Create()
{
    if (!Request.IsAuthenticated)
    {
        FormsAuthentication.RedirectToLoginPage();
    }
    return View();
}

public ActionResult Edit()
{
    if (!Request.IsAuthenticated)
    {
        FormsAuthentication.RedirectToLoginPage();
    }
    return View();
}
```

soit

```
[Authorize]
public ActionResult Index()
{
    return View();
}
[Authorize]
public ActionResult Create()
{
    return View();
}
[Authorize]
public ActionResult Edit()
{
    return View();
}
```

ou

```
[Authorize]
public class AdminController : Controller
{
    //
    // GET: /Admin/Admin/
    public ActionResult Index()
    {
        return View();
    }
    public ActionResult Create()
    {
        return View();
    }
    public ActionResult Edit()
    {
        return View();
    }
}
```

FILTRES

Un filtre est une annotation .NET qui ajoute une étape supplémentaire dans le process de requête.
Il existe en fait 4 filtres basiques :

Type	Interface	Implémentation par défaut	Description
Authorization	IAuthorizationFilter	AuthorizeAttribute	Exécuté avant tout autre filtre ou action
Action	IActionFilter	ActionFilterAttribute	Exécuté avant et après une action
Result	IResultFilter	ActionFilterAttribute	Exécuté avant et après que le résultat soit produit
Exception	IExceptionFilter	HandleErrorAttribute	Exécuté uniquement si un autre filtre, action, résult lève une exception

FILTRES

Les filtres peuvent être appliqués à chaque méthode ou à la totalité d'un Controller.

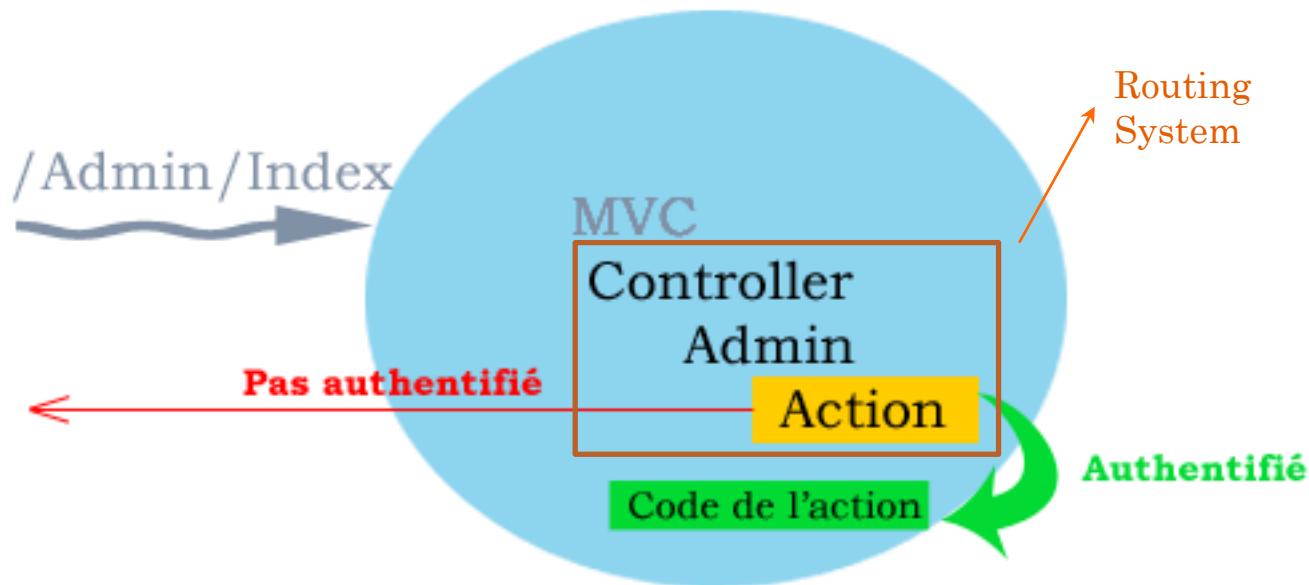
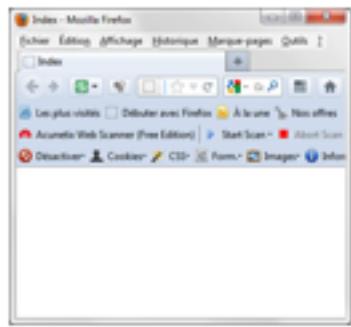
Il est possible de combiner les filtres.

```
[Authorize(Roles = "Administrator")] // tout le controller
public class ExampleController : Controller
{
    [OutputCache(Duration = 60)] // juste pour cette action
    public ActionResult Index()
    {
        return View();
    }
}
```

➤ Authorization Filters

Ce filtre force l'authentification et s'assure donc que les méthodes ne puissent être invoquée que par un utilisateur authentifié.

FILTRES



FILTRES

Pour permettre la vérification, nous avons à notre disposition les *AuthorizeAttributes*.

- Users (string) → les utilisateurs autorisés séparés par des ,
- Roles (string) → les rôles auxquels les users doivent appartenir



Il est également possible de créer son propre filtre d'autorisation.

(SampleAuthenticationFilter.sln)

- CustomAuthAttribute
- OrAuthorizationAttribute
- AjaxAuthorizeAttribute

FILTRES

- Exception Filters

MODEL TEMPLATES

209

MODEL TEMPLATES

Les *html helpers* vu précédemment (Html.CheckBoxFor, Html.TextBoxFor,...) permet de « générer » les éléments html requis pour éditer une partie des données.

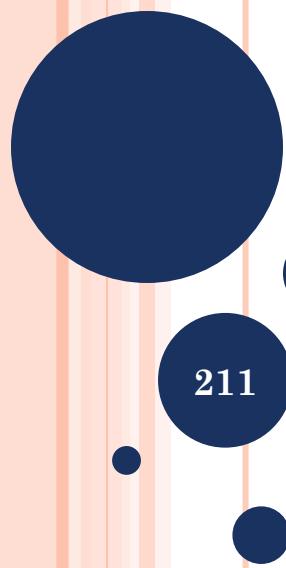
Il existe une alternative : *templated view helpers* qui permet de spécifier quel objet du model ou propriété nous désirons éditer, afficher... Ensuite c'est le framework qui déduit l'élément html à utiliser.

➤ L'utilisation

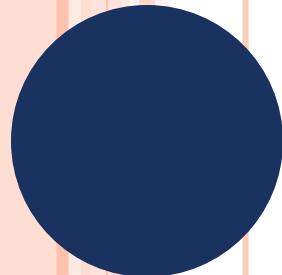
Le maître mot : **flexibilité**

Nous n'avons pas à nous inquiéter ni de la spécification html que nous voulons utiliser, ni de la mise à jour de la vue.

Voici un exemple :



MODEL BINDING



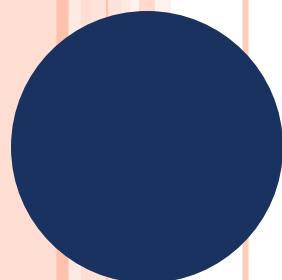
212



MODEL VALIDATION







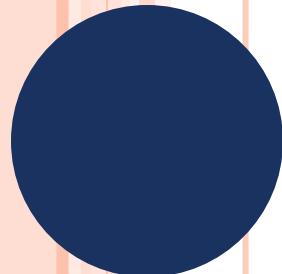
214

JQUERY



SÉCURITÉ

215



216

TOOLS



ASP.NET MVC 3 – TOOLS

- Ninject

NInject est un framework d'IOC.

Le principe d'Ioc (Inversion of control) permet de proposer à une application de multiples implémentations de ses composants métiers. Cette application ne sera pas impactée au niveau de son code si l'on décide de les modifier ou de les remplacer.

Avec Ioc, nous pouvons transporter le travail d'instanciation (`IBusiness monBusiness = new Business();`) en dehors de l'application.

Ou plus précisément, faire faire cette instanciation par un petit framework de façon dynamique, qui se basera sur un fichier de configuration dans lequel vous aurez répondu à la question :

Quelle interface utilise quelle classe pour s'instancier ?

ASP.NET MVC 3 - NINJECT

- Les avantages

Utiliser l'inversion de contrôle offre les avantages suivants :

- Chaque système ne se concentre que sur sa ou ses tâches principales.
- Les différents systèmes ne font pas d'hypothèses sur le comportement des autres systèmes.
- Remplacer un système ne produit pas d'effets de bord sur les autres systèmes, tant que le contrat d'origine est respecté.
- Dans le cas d'une nouvelle version d'un composant (ou d'un composant alternatif, comme un changement de Framework de log, par exemple), il est plus facile de changer le composant appelé.

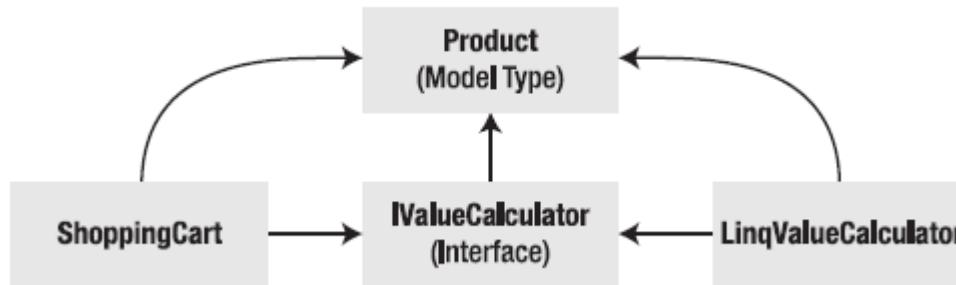
ASP.NET MVC 3 - NINJECT

- **Exemple**

Pour un panier, nous avons besoin de calculer le prix de chaque articles.

Nous avons donc :

- un model : Product
- un interface : IValueCalculator
- Une implémentation de l'interface : LinqValueCalculator
- Notre code pour le panier : ShoppingCart



ASP.NET MVC 3 - NINJECT

- Product.cs

```
public class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public decimal Price { get; set; }
    public string Category { set; get; }
}
```

- IValueCalculator

```
public interface IValueCalculator
{
    decimal ValueProducts(params Product[] products);
}
```

- LinqValueCalculator

```
public class LinqValueCalculator:IValueCalculator
{
    public decimal ValueProducts(params Product[] products)
    {
        return products.Sum(p => p.Price);
    }
}
```

ASP.NET MVC 3 - NINJECT

➤ ShoppingCart

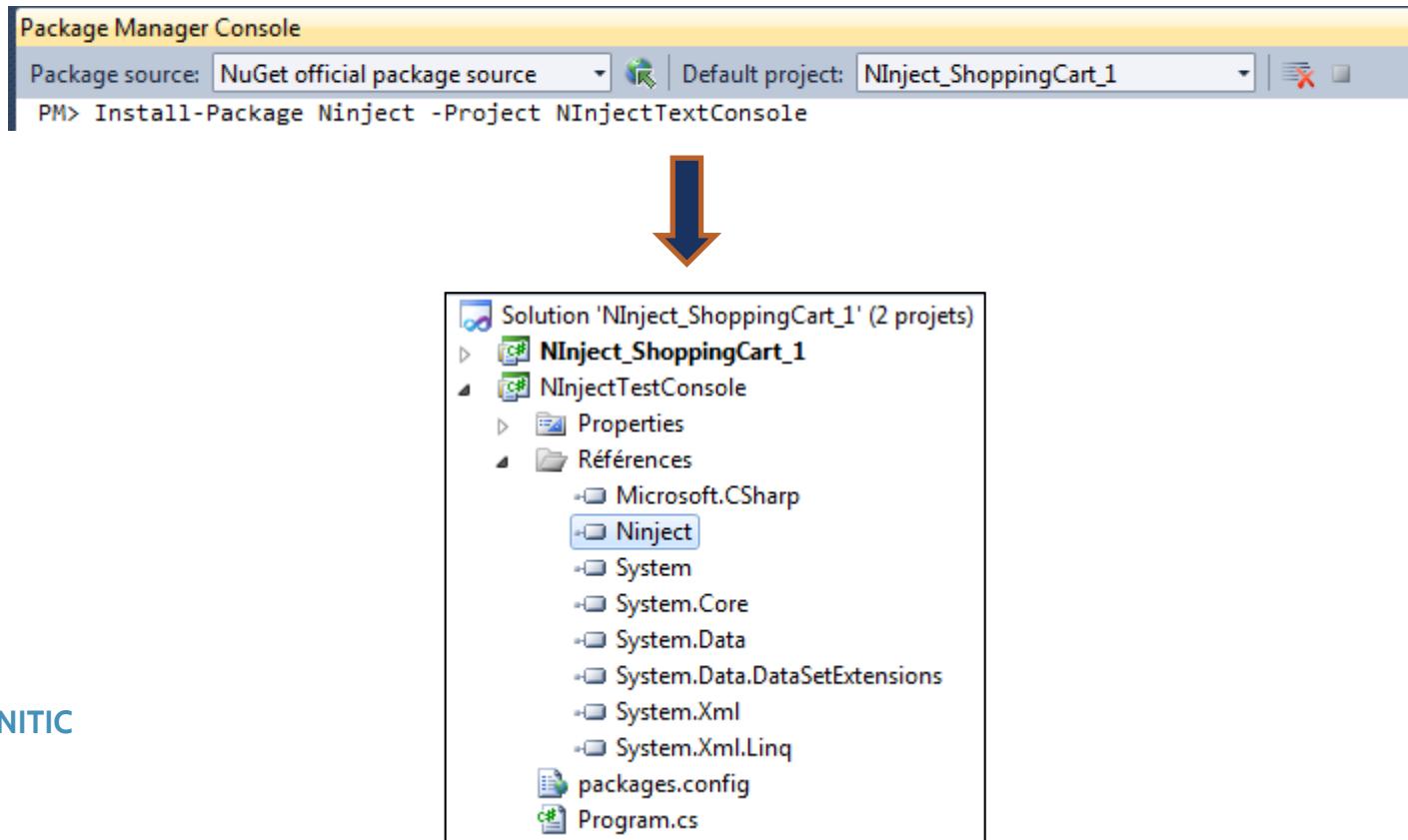
```
public class ShoppingCart
{
    private IValueCalculator calculator;
    public ShoppingCart(IValueCalculator calcParam)
    {
        calculator = calcParam;
    }
    public decimal CalculateStockValue()
    {
        // define the set of products to sum
        Product[] products = {
            new Product() { Name = "Kayak", Price = 275M},
            new Product() { Name = "Lifejacket", Price = 48.95M},
            new Product() { Name = "Soccer ball", Price = 19.50M},
            new Product() { Name = "Stadium", Price = 79500M}
        };
        // calculate the total value of the products
        decimal totalValue = calculator.ValueProducts(products);
        // return the result
        return totalValue;
    }
}
```

Le constructeur attend une implémentation de l'interface *IValueCalculator* qui sera préparé par notre DI. *ShoppingCart* et *LinqValueCalculator* ne se connaissent pas mais vont travailler ensemble grâce à l'interface *IValueCalculator*

ASP.NET MVC 3 - NINJECT

- Utilisation de Ninject

Il nous faut tout d'abord récupérer via Nuget le package



ASP.NET MVC 3 - NINJECT

- Préparation

Nous avons besoin d'un objet nous permettant de communiquer avec Ninject.

Pour cela, nous devons créer une instance du *Kernel*

```
class Program
{
    static void Main(string[] args)
    {
        IKernel ninjectKernel = new StandardKernel();
    }
}
```

Il y a deux façon de travailler avec Ninject :

- Lier le type que l'on désire avec l'interface créé
 - Dans notre exemple, Ninject recevra donc une demande d'implémentation de *IValueCalculator* et devra renvoyer une instance de *LinqValueCalculator*

```
ninjectKernel.Bind<IValueCalculator>().To<LinqValueCalculator>();
```

- Utiliser la méthode Ninject Get pour créer un objet qui implémente l'interface et le passe au constructeur de *notre classe ShoppingCart*

```
// Récupérer l'implémentation
IValueCalculator calcImpl = ninjectKernel.Get<IValueCalculator>();
// Création d'une instance de la classe ShoppingCart et injection de la dépendance
ShoppingCart cart = new ShoppingCart(calcImpl);
// Effectuer le calcul et écrire le résultat
Console.WriteLine("Total: {0:c}", cart.CalculateStockValue());
```

ASP.NET MVC 3 - NINJECT

- Création d'une chaîne de dépendance

Lorsque nous demandons à Ninject de créer une instance , il examine le couplage entre celui-ci et les autres types. S'il il existe des dépendance, il les résout pour nous et crée les instances de tous les types requis.

Exemple :

On crée une nouvelle interface *IDiscountHelper*

```
public interface IDiscountHelper
{
    decimal ApplyDiscount(decimal totalParam);
}
```

Nous créons une classe implémentant cet interface

```
public class DefaultDiscountHelper:IDiscountHelper
{
    public decimal ApplyDiscount(decimal totalParam)
    {
        return (totalParam - (10m / 100m * totalParam));
    }
}
```

ASP.NET MVC 3 - NINJECT

Et nous ajoutons une dépendance dans notre classe *LinqValueCalculator*

```
public class LinqValueCalculator:IValueCalculator
{
    private IDiscountHelper discounter;
    public LinqValueCalculator(IDiscountHelper discountParam)
    {
        discounter = discountParam;
    }
    public decimal ValueProducts(params Product[] products)
    {
        return discounter.ApplyDiscount(products.Sum(p => p.Price));
    }
}
```

Il nous reste à informer notre kernel de l'existence de cette nouvelle implémentation

```
static void Main(string[] args)
{
    IKernel ninjectKernel = new StandardKernel();

    ninjectKernel.Bind<IValueCalculator>().To<LinqValueCalculator>();
    ninjectKernel.Bind<IDiscountHelper>().To<DefaultDiscountHelper>();
    // Récupérer l'implémentation
    IValueCalculator calcImpl = ninjectKernel.Get<IValueCalculator>();

    // Création d'une instance de la classe ShoppingCart et injection de la dépendance
    ShoppingCart cart = new ShoppingCart(calcImpl);
    // Effectuer le calcul et écrire le résultat
    Console.WriteLine("Total: {0:c}", cart.CalculateStockValue());

    Console.ReadLine();
}
```

ASP.NET MVC 3 - NINJECT

- Les propriétés

Nous pouvons bien entendu configurer les classes que notre Kernel construit en lui fournissant les valeurs nécessaires pour les propriétés de celui-ci.

```
public class DefaultDiscountHelper : IDiscountHelper
{
    public decimal DiscountSize { get; set; }
    public decimal ApplyDiscount(decimal totalParam)
    {
        return (totalParam - (DiscountSize / 100m * totalParam));
    }
}
```

```
static void Main(string[] args)
{
    IKernel ninjectKernel = new StandardKernel();

    ninjectKernel.Bind<IValueCalculator>().To<LinqValueCalculator>();
    ninjectKernel.Bind<IDiscountHelper>().To<DefaultDiscountHelper>().WithPropertyValue("DiscountSize", 50M);
    // Récupérer l'implémentation
    IValueCalculator calcImpl = ninjectKernel.Get<IValueCalculator>();
```

Remarque : Le nom de la propriété est OBLIGATOIRE
COGNITIC

ASP.NET MVC 3 - NINJECT

- Le passage de paramètre au constructeur

Il est également possible de passer des paramètre au constructeur via notre kernel.

```
public class DefaultDiscountHelper : IDiscountHelper
{
    private decimal discountRate;
    public DefaultDiscountHelper(decimal discountParam)
    {
        discountRate = discountParam;
    }
    public decimal ApplyDiscount(decimal totalParam)
    {
        return (totalParam - (discountRate / 100m * totalParam));
    }
}
```

```
static void Main(string[] args)
{
    IKernel ninjectKernel = new StandardKernel();

    ninjectKernel.Bind<IValueCalculator>().To<LinqValueCalculator>();
    //ninjectKernel.Bind<IDiscountHelper>().To<DefaultDiscountHelper>().WithPropertyValue("DiscountSize", 50M);
    ninjectKernel.Bind<IDiscountHelper>().To<DefaultDiscountHelper>().WithConstructorArgument("discountParam", 50M);
    // Récupérer l'implémentation
    IValueCalculator calcImpl = ninjectKernel.Get<IValueCalculator>();

    // Création d'une instance de la classe ShoppingCart et injection de la dépendance
    ShoppingCart cart = new ShoppingCart(calcImpl);
```

ASP.NET MVC 3 - NINJECT

- Utilisation d'une classe dérivée

Nous pouvons également « binder » une classe dérivée à partir de sa classe mère.

```
public class LimitShoppingCart : ShoppingCart
{
    public LimitShoppingCart(IValueCalculator calcParam)
        : base(calcParam)
    {
        // nothing to do here
    }
    public override decimal CalculateStockValue()
    {
        // filter out any items that are over the limit
        var filteredProducts = products
            .Where(e => e.Price < ItemLimit);
        // perform the calculation
        return calculator.ValueProducts(filteredProducts.ToArray());
    }
    public decimal ItemLimit { get; set; }
}
```



```
IKernel ninjectKernel = new StandardKernel();
ninjectKernel.Bind<IValueCalculator>().To<LinqValueCalculator>();
ninjectKernel.Bind<IDiscountHelper>().To<DefaultDiscountHelper>().WithPropertyValue("DiscountSize", 50M);
ninjectKernel.Bind<LimitShoppingCart>().To<LimitShoppingCart>().WithPropertyValue("ItemLimit", 200M);
// Récupérer l'implémentation
IValueCalculator calcImpl = ninjectKernel.Get<IValueCalculator>();
// Création d'une instance de la classe ShoppingCart et injection de la dépendance
ShoppingCart cart = ninjectKernel.Get<LimitShoppingCart>();
```

ASP.NET MVC 3 - NINJECT

- Conditional Binding

Il est possible de prévoir plusieurs implémentations pour une même classe et donner à Ninject les instructions pour qu'il puisse choisir la bonne implémentation.

```
public class IterativeValueCalculator : IValueCalculator
{
    public decimal ValueProducts(params Product[] products)
    {
        decimal totalValue = 0;
        foreach (Product p in products)
        {
            totalValue += p.Price;
        }
        return totalValue;
    }
}
```

```
//Conditionnal
ninjectKernel.Bind<IValueCalculator>().To<IterativeValueCalculator>().WhenInjectedInto<LimitShoppingCart>();
// Récupérer l'implémentation
IValueCalculator calcImpl = ninjectKernel.Get<IValueCalculator>();
// Création d'une instance de la classe ShoppingCart et injection de la dépendence
ShoppingCart cart = ninjectKernel.Get<LimitShoppingCart>();
// Effectuer le calcul et écrire le résultat
Console.WriteLine("Total: {0:c}", cart.CalculateStockValue());
```

ASP.NET MVC 3 - NINJECT

Les différentes possibilités pour le *conditional binding*

Méthode	Effet
When(predicate)	Utilisé quand l'expression lambda = true
WhenClassHas<T>()	Utilisé quand la classe injectées est annoté avec le type T
WhenInjectedInto<T>()	Utilisé quand la classe injectées est de type T

ASP.NET MVC 3 - NINJECT

- Ninject et ASP.NET

La première chose à faire est de créer une classe qui sera utilisé par MVC pour instancier nos classes controllers.

Cette classe sera dérivée de *System.Web.Mvc.DefaultControllerFactory*

Elle va créer notre Kernel et utiliser le service pour la création des controller aux travers de la méthode *GetControllerInstance* appelée par MVC quand il veut un controller.

ASP.NET MVC 3 - NINJECT

```
public class NinjectControllerFactory : DefaultControllerFactory
{
    private IKernel ninjectKernel;
    public NinjectControllerFactory()
    {
        ninjectKernel = new StandardKernel();
        AddBindings();
    }
    protected override IController GetControllerInstance(RequestContext requestContext,
    Type controllerType)
    {
        return controllerType == null
            ? null
            : (IController)ninjectKernel.Get(controllerType);
    }
    private void AddBindings()
    {
        // put additional bindings here
        ninjectKernel.Bind<IPrductRepository>().To<FakeProductRepository>();
    }
}
```

Une fois cette classe créée, nous devons l'enregistrer pour que notre framework MVC puisse l'utiliser. Cela se fait dans le fichier Global.asax.

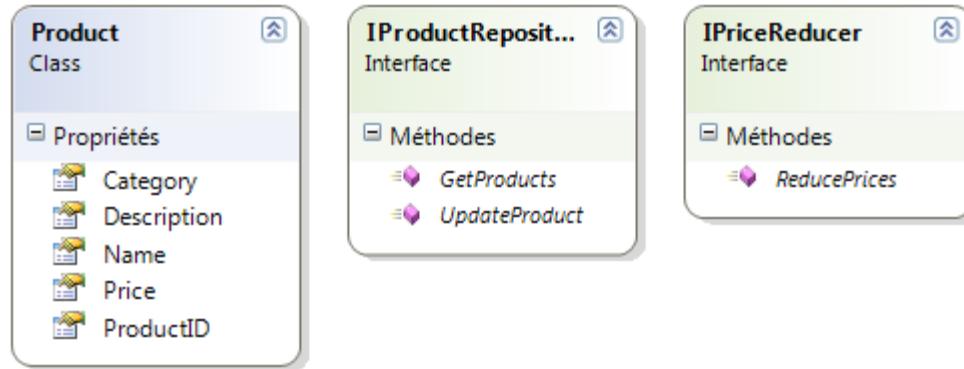
```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    RegisterGlobalFilters(GlobalFilters.Filters);
    RegisterRoutes(RouteTable.Routes);
    ControllerBuilder.Current.SetControllerFactory(new NinjectControllerFactory());
}
```

ASP.NET MVC 3 – UNIT TESTING

- Unit Testing avec VS2010

Même si il existe beaucoup de plugin pour les tests, il est tout à fait possible d'utiliser le support de test de notre VS2010.

La preuve par l'exemple :



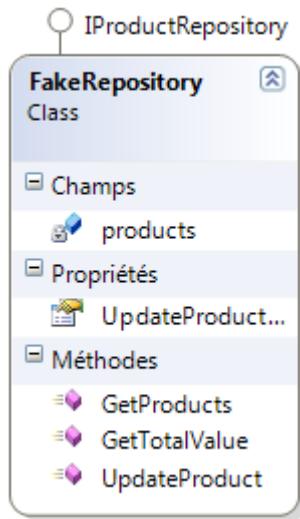
Le but est de créer une implémentation de l'interface *IPriceReducer* tel que :

Le prix de chaque produit dans le repository sera réduit

- Le total de la réduction soit égale à la valeur du paramètre *priceReduction * nbre de produits*
- *UpdateProduct* doit être appelé par tous les produits
- Aucun produit ne doit voir son prix réduit en dessous d'1 euro

ASP.NET MVC 3 – UNIT TESTING

Ensuite, nous créons la classe *FakeRepository* qui implémente l'interface *IProductRepository*.



```
public class FakeRepository : IProductRepository
{
    private Product[] products = {
        new Product() { Name = "Kayak", Price = 275M },
        new Product() { Name = "Lifejacket", Price = 48.95M },
        new Product() { Name = "Soccer ball", Price = 19.50M },
        new Product() { Name = "Stadium", Price = 79500M }
    };

    public IEnumerable<Product> GetProducts()
    {
        return products;
    }

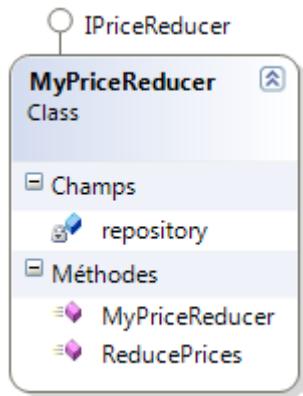
    public void UpdateProduct(Product productParam)
    {
        foreach (Product p in products
            .Where(e => e.Name == productParam.Name)
            .Select(e => e))
        {
            p.Price = productParam.Price;
        }
        UpdateProductCallCount++;
    }

    public int UpdateProductCallCount { get; set; }

    public decimal GetTotalValue()
    {
        return products.Sum(e => e.Price);
    }
}
```

ASP.NET MVC 3 – UNIT TESTING

Et pour terminer notre mise en place, nous créons une classe *MyPriceReducer* qui implémente l'interface *IPriceReducer*

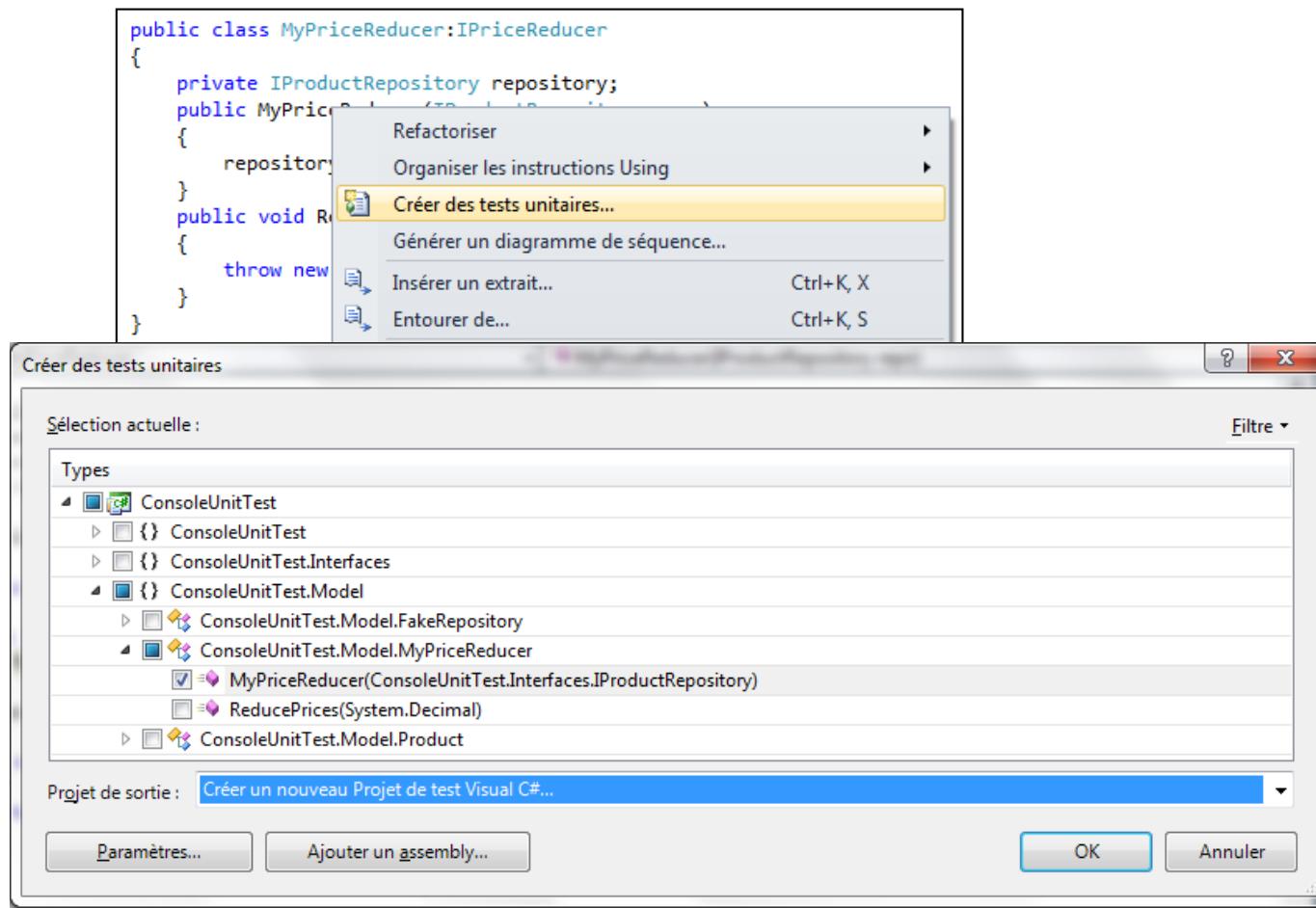


```
class MyPriceReducer:IPriceReducer
{
    private IProductRepository repository;
    public MyPriceReducer(IProductRepository repo)
    {
        repository = repo;
    }
    public void ReducePrices(decimal priceReduction)
    {
        throw new NotImplementedException();
    }
}
```

ASP.NET MVC 3 – UNIT TESTING

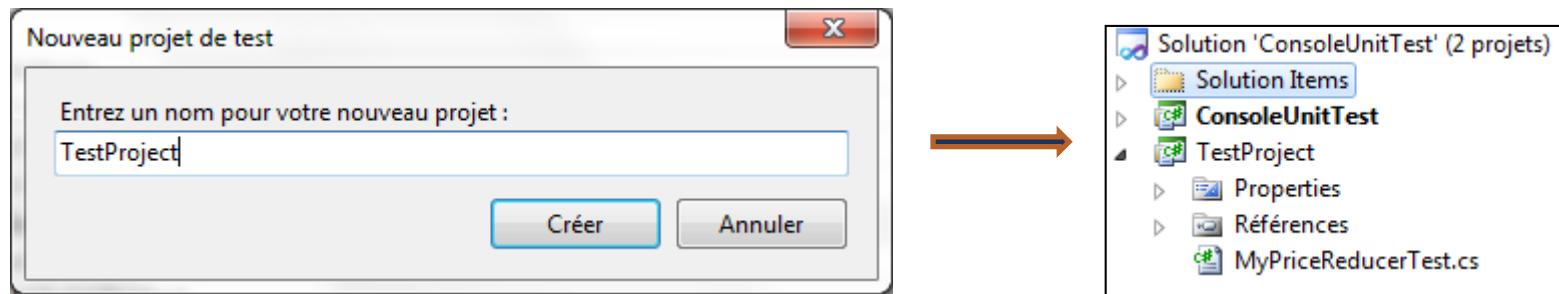
- Création des tests

La création d'un test unitaire se fait simplement via le menu contextuel



ASP.NET MVC 3 – UNIT TESTING

Le test unitaire est créé dans un projet extérieur par rapport à ce que nous avons sélectionné dans la boîte de dialogue.



Pour notre premier test, nous allons ignorer tout ce qui est suggéré dans notre classe de test afin d'implémenter les tests qui nous intéresse.

ASP.NET MVC 3 – UNIT TESTING

Nous utiliserons le pattern Arrange/Act/Assert pour notre méthode de test

```
[TestClass()]
public class MyPriceReducerTest
{
    [TestMethod]
    public void All_Prices_Are_Changed()
    {
        // Arrange
        FakeRepository repo = new FakeRepository();
        decimal reductionAmount = 10;
        IEnumerable<decimal> prices = repo.GetProducts().Select(e => e.Price);
        decimal[] initialPrices = prices.ToArray();
        MyPriceReducer target = new MyPriceReducer(repo);
        // Act
        target.ReducePrices(reductionAmount);
        prices = target(initialPrices, (p1, p2) =>
        {
            if (p1 == p2)
            {
                Assert.Fail();
            }
            return p1;
        });
    }
}
```

être sur que tous
ont été changés

ASP.NET MVC 3 – UNIT TESTING

Notre seconde méthode de test

```
[TestMethod]
public void No_Price_Less_Than_One_Dollar()
{
    // Arrange
    FakeRepository repo = new FakeRepository();
    decimal reductionAmount = decimal.MaxValue;
    MyPriceReducer target = new MyPriceReducer(repo);
    // Act
    target.ReducePrices(reductionAmount);
    // Assert
    foreach (Product prod in repo.GetProducts())
    {
        Assert.IsTrue(prod.Price >= 1);
    }
}
```

Nous suivons le même pattern A/A/A pour notre seconde méthode.

Nous utilisons également *Assert.IsTrue* qui permet de renvoyer le résultat booléen de l'expression lambda. Si False, le test a échoué.

ASP.NET MVC 3 – UNIT TESTING

Méthode	Description
AreEqual<T>(T, T) AreEqual<T>(T, T, string)	Vérifie que les deux <T> ont la même valeur
AreNotEqual<T>(T, T) AreNotEqual<T>(T, T, string)	Vérifie que les deux <T> n'ont pas la même valeur
AreSame<T>(T, T) AreSame<T>(T, T, string)	Vérifie que les deux variables pointent sur le même objet
AreNotSame<T>(T, T) AreNotSame<T>(T, T, string)	Vérifie que les deux variables ne pointent pas sur le même objet
Fail() Fail(string)	Echec
Inconclusive() Inconclusive(string)	Indique que le résultat du test est incertain
IsTrue(bool) IsTrue(bool, string)	Évalue l'expression booléenne et renvoi le résultat
IsFalse(bool) IsFalse(bool, string)	Évalue l'expression booléenne et renvoi l'opposé du résultat
IsNull(object) IsNull(object, string)	Vérifie si l'objet est null

ASP.NET MVC 3 – UNIT TESTING

Méthode	Description
IsNotNull(object) IsNotNull(object, string)	Vérifie que l'objet n'est pas null
IsInstanceOfType(object, Type) IsInstanceOfType(object, Type, string)	Vérifie si l'objet est une instance de <T>
IsNotInstanceOfType(object, Type) IsNotInstanceOfType(object, Type, string)	Vérifie si l'objet n'est pas une instance de <T>

C'est méthodes static permettent de vérifier certains Aspect des tests. Si une *Assertion* échoue, une exception est lancée et fait échouer le test.

Chaque test est exécuté séparément si un échoue, les autres continueront à fonctionner.

Le string inclus dans les signatures permet d'envoyer un message vers l'exception afin de mieux tracer les erreurs.

ASP.NET MVC 3 – UNIT TESTING

Résultats des tests			
Résultat	Nom du test	Projet	Message d'erreur
Échec	All_Prices_Are_Changed	TestProject	La méthode de test TestProject.MyPriceReducerTest.All_Prices_Are_Changed() n'a pas été implémentée.
Échec	No_Price_Less_Than_One_Dollar	TestProject	La méthode de test TestProject.MyPriceReducerTest.No_Price_Less_Than_One_Dollar() n'a pas été implémentée.



Normal car nous n'avons pas implémenté la méthode ReducePrice dans notre classe *MyPriceReducer*



```
public void ReducePrices(decimal priceReduction)
{
    foreach (Product p in repository.GetProducts())
    {
        p.Price = Math.Max(p.Price - priceReduction, 1);
        repository.UpdateProduct(p);
    }
}
```



Résultats des tests			
Résultat	Nom du test	Projet	Message d'erreur
Réussite	All_Prices_Are_Changed	TestProject	
Réussite	No_Price_Less_Than_One_Dollar	TestProject	

ASP.NET MVC 3 – MoQ

Nous avons mis en place la classe *FakeRepository* pour simuler l'utilisation de notre *IProductRepository*.

Cette classe est appelée une implémentation *Mock* de notre *IProductRepository*.

Nous avons en fait mis en place uniquement ce qui était nécessaire pour notre test unitaire.

Grâce au Framework *Moq*, nous pouvons facilement mettre en place des implémentations *Mock* pour faciliter nos tests unitaires.

les *mocks* (ou *Mock object*) sont des objets simulés qui reproduisent le comportement d'objets réels de manière contrôlée

- **La création d'un Mock avec Moq**

L'avantage d'utiliser Moq est de permettre de « générer » nos Mock avec juste ce qu'il faut pour nos tests.

Afin de mettre en place nos Mock, nous devons passer par deux étapes :

ASP.NET MVC 3 – MoQ

1. Création de Mock<T>

T sera le type pour lequel nous désirons créer notre Mock.

```
Mock<IPProductRepository> mock = new Mock<IPProductRepository>();
```

2. Configuration du Mock

Moq implémente automatiquement tous les propriétés, méthodes du type que nous lui avons passé dans notre instanciation.

Pour cela, il utilise les valeurs par défaut ! Par exemple, *IPProductRepository.GetProducts* nous retournera un *Ienumerable<Product>* vide.

Pour changer ce comportement, nous allons utiliser la méthode *Setup* pour configurer notre behavior

```
public FakeRepository()
{
    Product[] products = new Product[] {
        new Product() { Name = "Kayak", Price = 275M },
        new Product() { Name = "Lifejacket", Price = 48.95M },
        new Product() { Name = "Soccer ball", Price = 19.50M },
        new Product() { Name = "Stadium", Price = 79500M }
    };

    Mock<IPProductRepository> mock = new Mock<IPProductRepository>();
    mock.Setup(m => m.GetProducts()).Returns(products);
}
```

ASP.NET MVC 3 – MoQ

Il y a 3 éléments à considérer pour la configuration

A. Utiliser le select

Moq utilise la syntaxe Linq et les expressions lambda.

Donc si nous désirons que Moq utilise notre *GetProduct* nous écrirons

```
mock.Setup(m => m.GetProducts())
```

B. Utiliser les filtres

Nous pouvons également demander à Moq de régir suivant des paramètres que nous lui passerons

```
public interface IMyInterface  
{  
    string ProcessMessage(string message);  
}
```

```
Mock<IMyInterface> mock2 = new Mock<IMyInterface>();  
mock2.Setup(m => m.ProcessMessage("Bonjour")).Returns("Salut");  
mock2.Setup(m => m.ProcessMessage("Aurevoir")).Returns("A bientôt");
```

Moq interprète ces instructions pour retourner le message correspondant au paramètre. Pour tout autre valeur, Moq renverra la valeur par défaut.

Nous pouvons également cibler un type de paramètre grâce à la méthode *It*

```
mock2.Setup(m => m.ProcessMessage(It.IsAny<string>())).Returns("Message received");
```

ASP.NET MVC 3 – MoQ

Méthode	Description
Is<T>()	Match si le prédicat renseigné est vérifié
IsAny<T>()	Match si n'importe quel paramètre est de type T
IsInRange<T>()	Match si le paramètre est entre les bornes
IsRegex	Match si le paramètre est vérifié par l'expression régulière renseignée

Is<T> est la méthode la plus flexible puisque nous pouvons spécifier le prédicat qui nous intéresse

```
mock2.Setup(m => m.ProcessMessage(It.IsAny<string>(s => s == "hello" || s == "bye")))
    .Returns("Message received");
```

Moq nous retournera *Message received* si le paramètre est *hello* ou *bye*

ASP.NET MVC 3 – MoQ

C. Retourner un résultat

Lorsque nous mettons en place un Behavior, nous désirons généralement définir ce que retourne la méthode lorsqu'elle sera appelée.

Spécifier un retour est simple

```
Mock<IMyInterface> mock2 = new Mock<IMyInterface>();
mock2.Setup(m => m.ProcessMessage("Bonjour")).Returns("Salut");
mock2.Setup(m => m.ProcessMessage("Aurevoir")).Returns("A bientôt");
```

Mais nous pouvons également basé notre retour sur le paramètre via la méthode *Result<T>*

```
mock2.Setup(m => m.ProcessMessage(It.IsAny<string>()))
    .Returns<string>(s => string.Format("Message received: {0}", s));
```

ASP.NET MVC 3 – MoQ

- Moq et les test unitaires

Mettre en place des tests unitaire devient très simple.

Il nous suffit de configurer notre *Mock*, lui attribuer filtre et prédictat...

Et ensuite, via la propriété *Object*, tester notre code.

```
[TestMethod]
public void Correct_Total_Reduction_Amount()
{
    // Arrange
    Product[] products = new Product[] {
        new Product() { Name = "Kayak", Price = 275M},
        new Product() { Name = "Lifejacket", Price = 48.95M},
        new Product() { Name = "Soccer ball", Price = 19.50M},
        new Product() { Name = "Stadium", Price = 79500M}
    };
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    mock.Setup(m => m.GetProducts()).Returns(products);
    decimal reductionAmount = 10;
    decimal initialTotal = products.Sum(p => p.Price);
    MyPriceReducer target = new MyPriceReducer(mock.Object);
    // Act
    target.ReducePrices(reductionAmount);
    // Assert
    Assert.AreEqual(products.Sum(p => p.Price),
        (initialTotal - (products.Count() * reductionAmount)));
}
```

ASP.NET MVC 3 – MoQ

Il est possible de faciliter encore le processus de test.

En effet, nous savons que toutes nos méthodes de tests vont travailler sur le même ensemble de produits... Nous pouvons donc l'intégrer dans notre classe de test

```
[TestClass()]
public class MyPriceReducerTest
{
    private IEnumerable<Product> products;
    [TestInitialize]
    public void PreTestInitialize()
    {
        products = new Product[] {
            new Product() { Name = "Kayak", Price = 275M },
            new Product() { Name = "Lifejacket", Price = 48.95M },
            new Product() { Name = "Soccer ball", Price = 19.50M },
            new Product() { Name = "Stadium", Price = 79500M }
        };
    }
}
```

Comme nous désirons (re-)commencer nos tests avec des données « clean », nous utilisons une méthode pour initialiser notre set de données.

VS2010 cherche une méthode avec l'annotation *[TestInitialize]* et si il en trouve une, il l'utilisera²⁴⁹ avant chaque test.

ASP.NET MVC 3 – MoQ

Annotations	Description
ClassInitialize	Appelé avant que la classe de test ne soit utilisée. (Doit être appliqué sur une méthode Static)
ClassCleanup	Appelé après que tous les tests aient été effectués (Doit être appliqué sur une méthode Static)
TestInitialize	Appelé avant chaque exécution de test
TestCleanup	Appelé après chaque exécution de test

Le nom de la méthode annotée n'a pas d'importance, Visual Studio ne s'occupe que de l'annotation

ASP.NET MVC 3 – MoQ

- La vérification avec Moq

L'un de nos critère est : la méthode *UpdateProduct* doit être appelée pour chaque produit

Comment vérifier cela ?

```
// Act  
target.ReducePrices(reductionAmount);  
// Assert  
foreach (Product p in products)  
{  
    mock.Verify(m => m.UpdateProduct(p), Times.Once());  
}
```

Avec la méthode *Verify*, nous pouvons vérifier la fréquence et dans notre cas, être sur que notre *UpdateProduct* n'est appelée qu'1! fois par produit

