

# ASP.Net WebForms

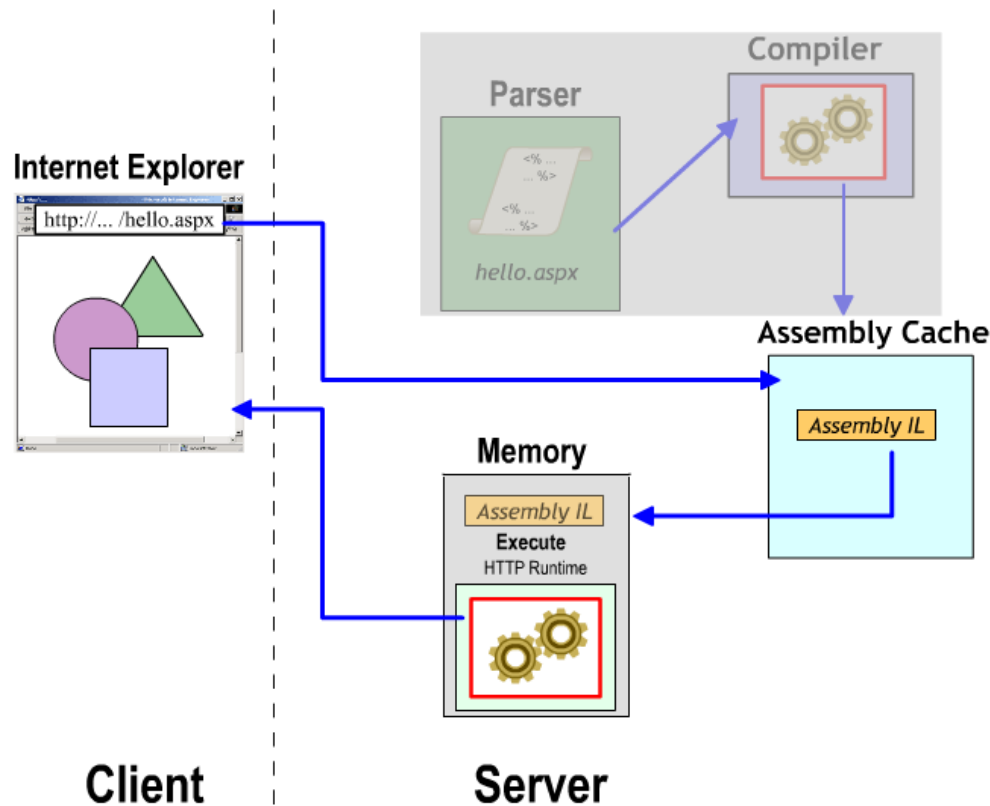
Préparé par  
M.G. BELKASMI

# ASP.Net

- ASP.Net permet la programmation d'applications Web dynamiques.
- Les navigateurs Web, à l'aide de pages au format html, servent d'interface entre l'application .Net et l'utilisateur
- Contrairement à ASP, où le code était inclus directement dans la partie html, ASP.Net est un langage compilé
- La partie html (interface) et la partie C# (traitements) peuvent ainsi être séparées au sein d'un même fichier ou sur deux fichiers distincts

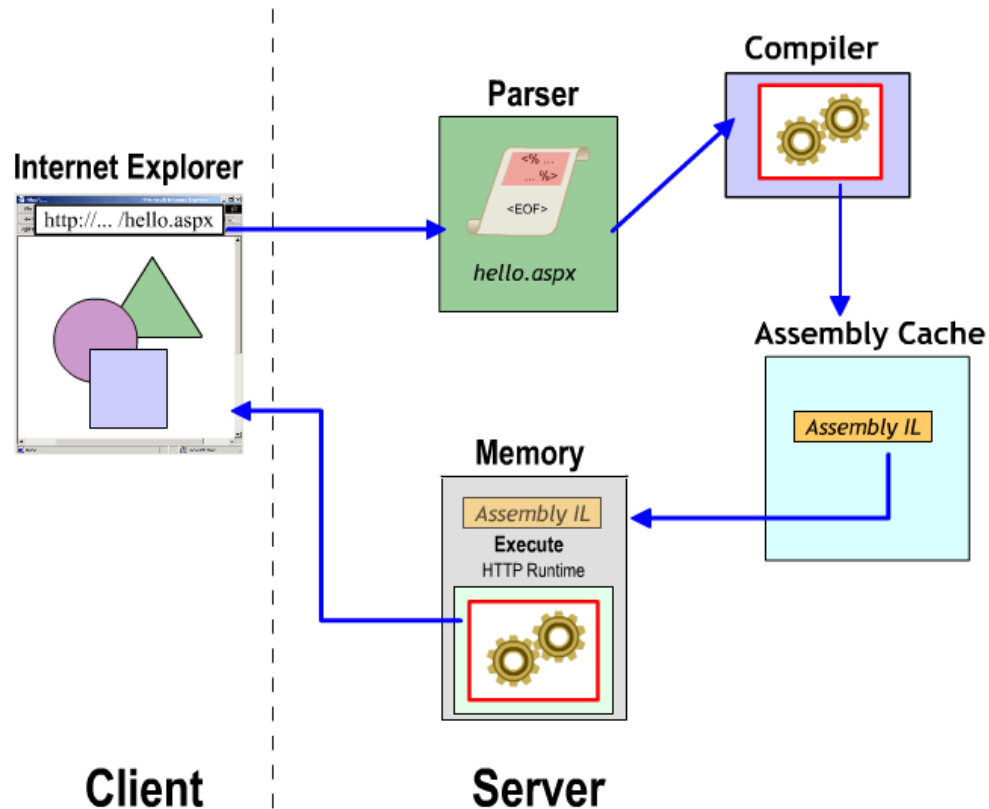
# ASP.Net

- Lors de la réception d'une demande venant d'un navigateur, le serveur vérifie si la page a déjà été compilée
- Si c'est le cas, la page, déjà au format html est envoyée au navigateur du client.



# ASP.Net

- Dans le cas contraire, la page au format ASP.Net est d'abord compilée au format M.S.I.L ensuite, la page est générée au format html. Cette page peut contenir des scripts utilisant des langages utilisés du côté du client.



# ASP.Net

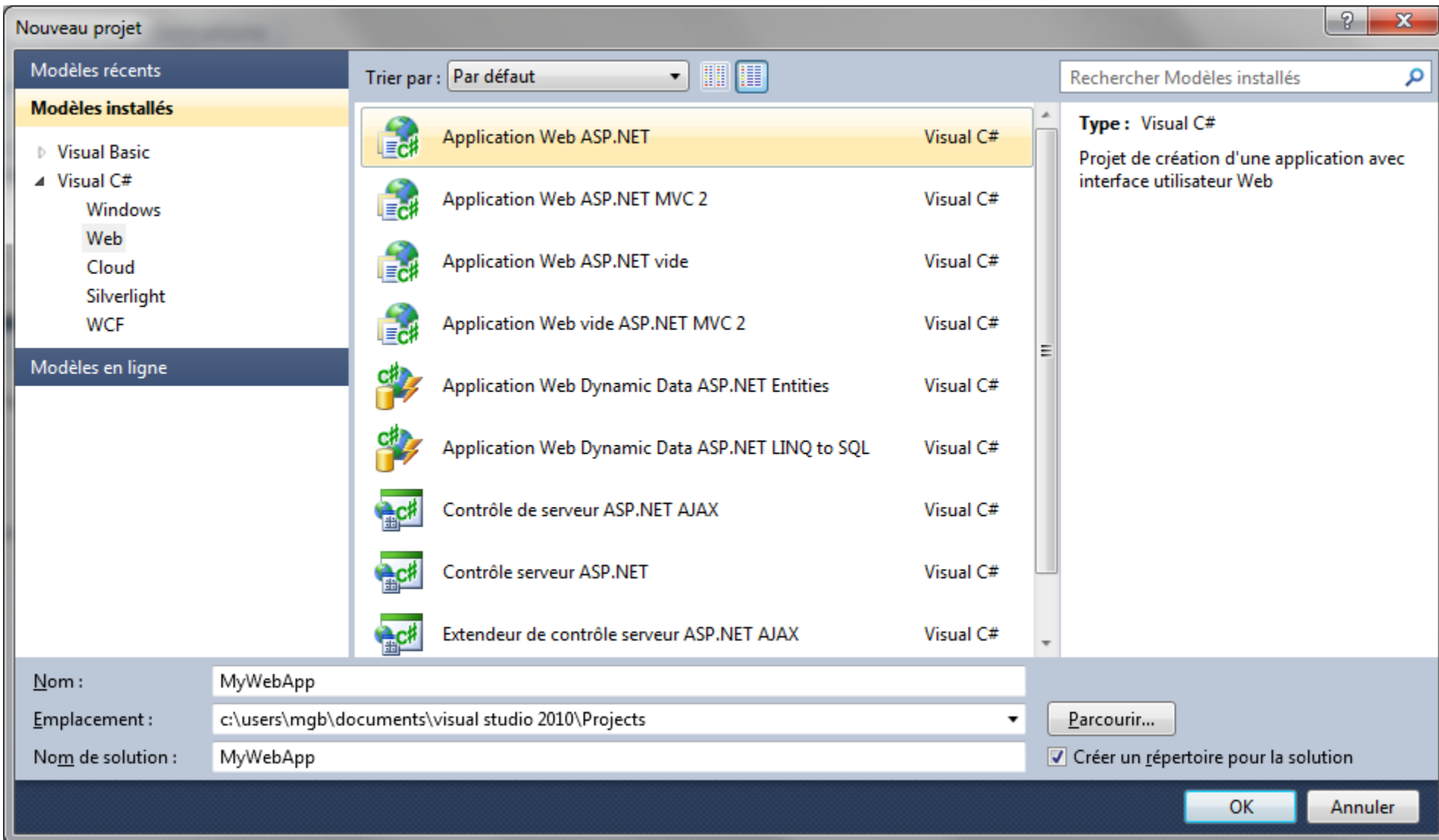
- L'ASP.Net permet la séparation des couches. Cette méthode est très employée en entreprise pour permettre une meilleure organisation du code. Il est ainsi plus facile de le faire évoluer, de le maintenir et de le corriger.
- Cela consiste à isoler dans des dossiers / fichiers les couches, on appelle cela l'architecture n-tiers.

# ASP.Net

## Présentation d'un projet ASP.Net

- Une fois dans Visual Studio, faites Fichier > Nouveau > Projet.
- Choisir son langage (VB.NET ou C#) et cliquer sur Web.
- Vérifier que « Application Web ASP.Net » est bien sélectionné.
- Entrer un nom de projet, choisir son emplacement et le nom de la solution (généralement on met le même nom de solution que celui du projet).
- Cliquer sur « Ok » en bas à droite

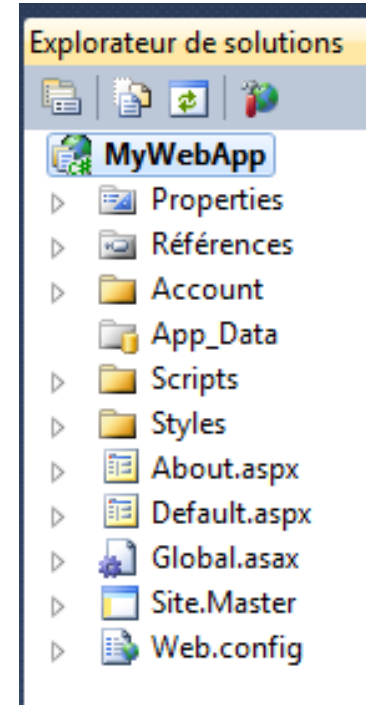
# ASP.Net



# ASP.Net

## Dossiers et fichiers du projet

- **Properties** : permet la configuration, de l'application Web ASP.Net et du Visual Studio, propre à ce projet.
- **Références** : Ensemble des Références que l'on va utiliser dans son Application Web ASP.NET.
- **Account** : Dossier avec des pages créées par défaut pour la gestion de l'authentification
- **App\_Data** : contient les fichiers des données d'application (fichiers de BD, fichiers XML, ...)
- **Scripts** : on peut y placer les fichiers JavaScript. Par défaut, JQuery 1.4.1 est présent.
- **Styles** : On y place les fichiers CSS.

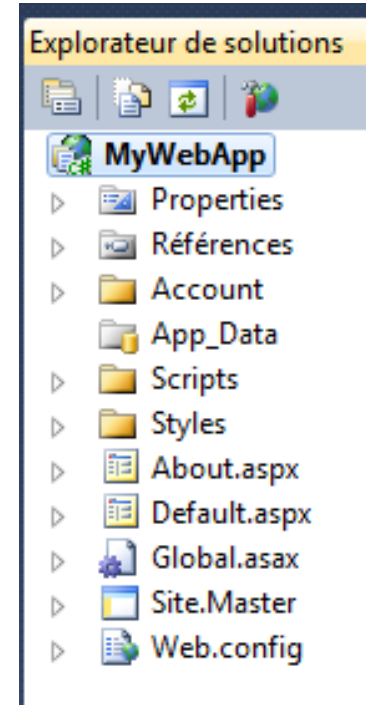




# ASP.Net

## Dossiers et fichiers du projet

- **About.aspx** et **Default.aspx** sont des fichiers créés par défaut. Sur IIS, le fichier appelé par défaut dans un dossier est default.aspx.
- **Global.asax** : fichier facultatif qui contient du code pour répondre aux événements de niveau application ou session déclenchés par ASP.Net ou par les modules HTTP
- **Site.Master** : Master page. Définit la disposition des autres pages Web de l'application.
- **Web.Config** : Fichier de paramétrage de l'application



# ASP.Net

## Types de fichiers du projet Web ASP.Net

Type de fichier	Description
.aspx	En général, un fichier Global.aspx qui représente la classe d'application et contient des méthodes facultatives (gestionnaires d'événements) s'exécutant à différents points du cycle de vie de l'application.
.ascx	Un fichier de contrôle utilisateur Web qui définit des fonctionnalités personnalisées susceptibles d'être ajoutées à toutes les pages Web Forms ASP.Net
.asmx	Fichier de services Web XML qui contient des classes et des méthodes susceptibles d'être appelées par d'autres applications Web.
.aspx	Page Web Forms ASP.Net qui peut contenir des contrôles Web, ainsi que la logique métier et de présentation
.cs, .vb	Les fichiers de code source (fichiers .cs ou .vb) qui définissent le code pouvant être partagé entre les pages, par exemple le code des classes personnalisées

# ASP.Net

## Types de fichiers du projet Web ASP.Net

Type de fichier	Description
.dll	Fichier de bibliothèque de classes compilé (assembly).
.master	Page maître qui définit la disposition des autres pages Web de l'application.
.mdb, .ldb	Fichier de base de données
.sitemap	Fichier sitemap qui définit la structure logique de l'application Web

# ASP.Net

## Les en-têtes / directives

- Il s'agit d'une ligne qui définit certains paramètres :

Default.aspx X

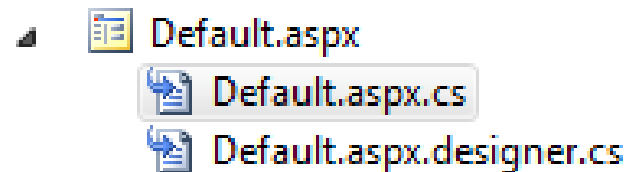
```
<%@ Page Title="Page d'accueil" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"  
CodeBehind="Default.aspx.cs" Inherits="MyWebApp._Default" %>
```

- Il y est indiqué que nous allons utiliser le C# comme langage (Language="C#"), que le nom de la page du code behind, correspondant à cette page .aspx, s'appelle Default.aspx.cs (CodeBehind="Default.aspx.cs").
- Est aussi définit le namespace et le nom de la classe lié à la page (Inherits="MyWebApp.\_Default").

# ASP.Net

## Le code Behind

- C'est le code dans lequel on écrira les instructions qui seront exécutées sur le serveur comme, par exemple, récupérer une valeur, changer un attribut dynamiquement, créer un objet,...
- On remarque dans l'explorateur de solution qu'une page web par défaut a été créée (Default.aspx) et que si l'on déroule l'arborescence il y a deux autres pages qui y sont liées :
  - La page default.aspx.cs
  - et
  - le designer

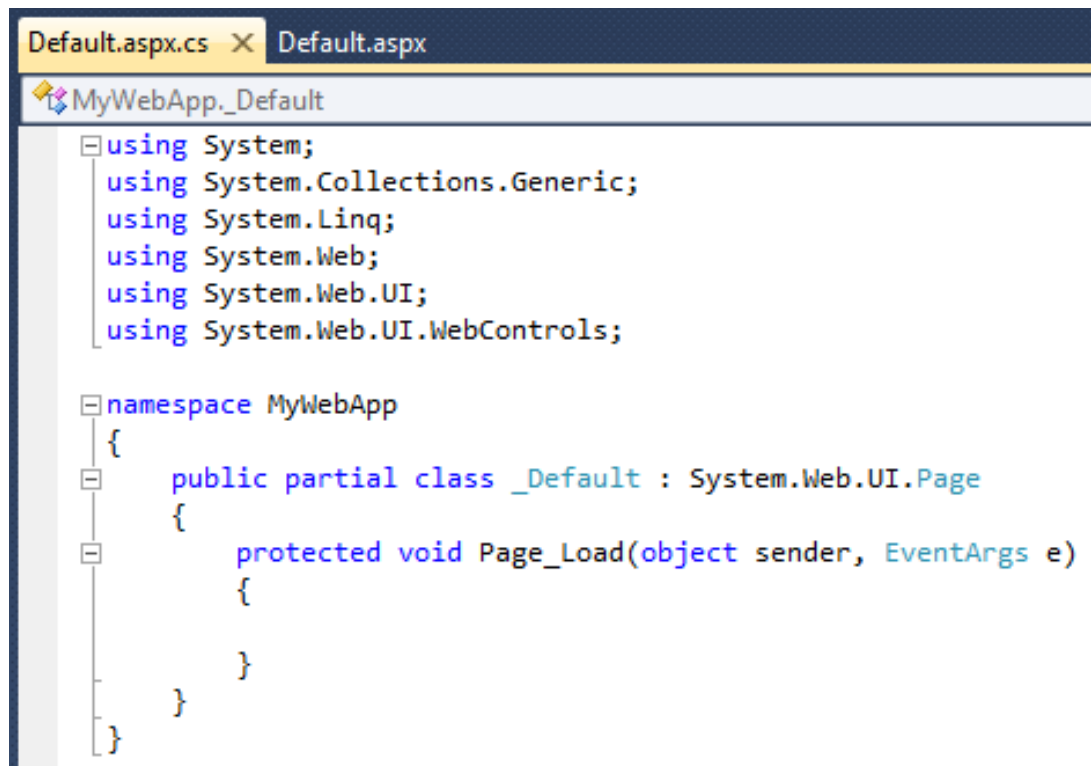


Rq: faire un clic droit sur la page aspx et faire "Afficher le code" pour accéder à Default.aspx.cs).

# ASP.Net

## Le code Behind

- Default.aspx.designer.cs contient du code behind généré automatiquement
- Default.aspx.cs contient le code behind qu'on ajoutera manuellement

A screenshot of a Visual Studio code editor window. The title bar shows two tabs: 'Default.aspx.cs' (active) and 'Default.aspx'. Below the tabs, the file path 'MyWebApp\_Default' is visible. The code is written in C# and includes several 'using' statements for System, System.Collections.Generic, System.Linq, System.Web, System.Web.UI, and System.Web.UI.WebControls. It defines a namespace 'MyWebApp' containing a partial class '\_Default' that inherits from 'System.Web.UI.Page'. The class has a 'Page\_Load' method with a 'sender' parameter and an 'EventArgs' parameter 'e'.

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

namespace MyWebApp
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
    }
}
```

# ASP.Net

## Rôle du client et du serveur

- Le rôle d'un navigateur est de récupérer et de traiter les données envoyées par le serveur. Il va exécuter le code qui doit l'être comme le Javascript et va afficher le résultat à l'écran de l'utilisateur.
- Le serveur reçoit / récupère les requêtes qui lui sont envoyées. Il va ensuite les traiter : la requête peut être la demande d'affichage d'une page ou l'envoi de données tels que celles d'un formulaire, il va exécuter la page ASPX correspondante et ensuite retourner le résultat. Il est important de savoir que ce que retourne le serveur ne contient pas d'ASP.Net mais n'est constitué que de HTML, CSS, Javascript ...

# ASP.Net

## HyperText Transfer Protocol (rappel)

- HTTP est un protocole de communication textuel utilisé entre les navigateurs et les serveurs. Il utilise le port 80
- Les méthodes HTTP sont les méthodes utilisées pour passer les données entre le serveur et le navigateur:
  - Head : contient des informations utiles (navigateur utilisé, page visitée, système d'exploitation, ...)
  - Get: Permet d'envoyer des données par l'adresse url (en clair)
  - Post : Permet d'envoyer des données par les http Headers



# ASP.Net

## Code Behind: les variables disponibles

- héritant de la classe Page, nous disposons d'un certains nombres de variables bien utiles.
- **Request**: objet centralisant les données de la requête effectuée par le navigateur
- **Response**: pendant de request, centralise les éléments qui seront sérialisés en HTML et qui seront renvoyés au navigateur
- **IsPostBack**: booléen permettant de savoir si la requête en cours émane de la page elle-même.
- **Session**: Cache utilisateur. Il s'agit d'un dictionnaire <string,object> permettant de stocker des informations pour l'utilisateur en cours.
- **Application**: Cache applicatif. Il fonctionne exactement comme la session. La seule différence est que les informations sont partagées entre tous les utilisateurs de l'application.

# ASP.Net

ASP.Net: un langage de balises

- Une page aspx n'est rien d'autre qu'une page html contenant du code pouvant être exécuté sur le serveur.
- On distingue 2 types de balises:
  - `<% code %>` : le code (C# ou VB) situé entre ces balises sera interprété au moment du render
  - `<asp:button>`, `<asp:label>` : les composants présentés sous cette forme seront analysés pendant l'init et seront donc utilisable au moment du load.
    - Ces appels ne sont en fait qu'une façon d'appeler du code objet au sein d'une page aspx.
    - On pourrait appeler ces objets directement depuis le code behind.

# ASP.Net

## ASP.Net: un langage de balises

- L'attribut `runat="server"`:
  - Cet attribut permet de dire au compilateur et au générateur de code qu'un composant doit être visible pour le serveur et plus particulièrement pour l'interpréteur .NET.
  - Sans cet attribut, la méthode d'init n'ajoutera pas le contrôle à la liste des contrôles accessibles dans le code behind.
  - Le fait de placer un `runat=server` sur n'importe quel élément de l'aspx (composant asp ou html) le fera apparaître dans le designer.

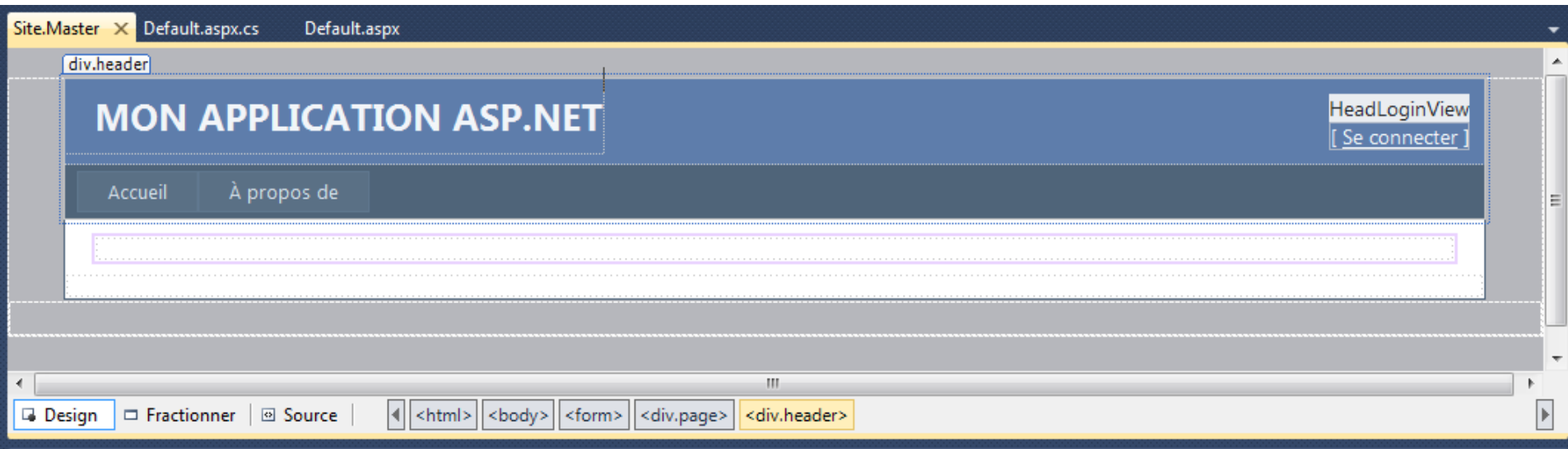
# ASP.Net

## Pages maîtres ASP.Net

- Une page maître est un fichier ASP.Net possédant l'extension .master (par exemple, MySite.master) avec une disposition prédéfinie pouvant inclure du texte statique, des éléments HTML et des contrôles serveur.
- Les pages maîtres ASP.Net permettent de créer une disposition cohérente des pages de l'application.
- Une page maître unique définit l'apparence et le comportement standard qu'on souhaite avoir pour toutes les pages (ou groupe de pages) de l'application

# ASP.Net

## Pages maîtres ASP.Net



# ASP.Net

## Pages maîtres ASP.Net

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.cs" Inherits="MyWebApp.SiteMaster" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head runat="server">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <title></title>
  <link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />
  <asp:ContentPlaceHolder ID="HeadContent" runat="server">
  </asp:ContentPlaceHolder>
</head>
<body>
  <form runat="server">
    <div class="page">
      <div class="header">
        <div class="title">
          <h1>
            Mon application ASP.NET
          </h1>
        </div>
        <div class="loginDisplay">
          <asp:LoginView ID="HeadLoginView" runat="server" EnableViewState="false">
            <AnonymousTemplate>

```

100 %

Design Fractionner Source

<html> <body> <form> <div.page> <div.header>

# ASP.Net

## Pages de contenu

- Définit le contenu des contrôles réservés de la page maître en créant des pages de contenu individuelles qui sont des pages ASP.Net (fichiers .aspx et, éventuellement, des fichiers code-behind) liées à une page maître spécifique.

```
<% @ Page Language="C#" MasterPageFile="~/Master.master" Title="Content Page 1" %>
<asp:Content ID="Content1" ContentPlaceHolderID="Main" Runat="Server">
    Main content.
</asp:Content>
```

```
<asp:Content ID="Content2" ContentPlaceHolderID="Footer" Runat="Server" >
    Footer content.
</asp:content>
```

# ASP.Net

## Pages de contenu

Référencement entre la master Page et la page de contenu:

- 1-Création d'une propriété publique dans le « code-behind » de la Master page
- 2-Les pages de contenu peuvent accéder à n'importe quelle propriété publique déclarée dans la Master Page.

```
8  
9 public String SharedInfo  
10 {  
11     get { return (String)Session["SharedInfo"]; }  
12     set { Session["SharedInfo"] = value; }  
13 }  
14
```

- Accès direct à la propriété dans le « code-behind » de la page de contenu

```
21  
22 infoLabel.Text = Master.SharedInfo;  
23
```



# ASP.Net

## Principes du contrôle serveur

- Un contrôle serveur est une balise (HTML ou ASP) associé à la propriété `runat="server"`.
- A cette balise sera associé un objet lors de l'exécution du code par le serveur, il sera donc instancié au chargement de la page et peut être appelé par le code behind grâce à son id
- Un contrôle serveur est programmable depuis le code behind pour répondre à des évènements (comme un clic, le chargement du contrôle et d'autres).

# ASP.Net

## Principes du contrôle serveur

- HTML Controls

Les contrôles HTML sont des contrôles serveurs dans le cas où ils ont un attribut `runat="server"`

### ▲ Hiérarchie d'héritage

```
System.Object
System.Web.UI.Control
System.Web.UI.HtmlControls.HtmlControl
System.Web.UI.HtmlControls.HtmlArea
System.Web.UI.HtmlControls.HtmlContainerControl
System.Web.UI.HtmlControls.HtmlImage
System.Web.UI.HtmlControls.HtmlInputControl
System.Web.UI.HtmlControls.HtmlLink
System.Web.UI.HtmlControls.HtmlMeta
System.Web.UI.HtmlControls.HtmlSource
System.Web.UI.HtmlControls.HtmlTitle
System.Web.UI.HtmlControls.HtmlTrack
```

# ASP.Net

### Principes du contrôle serveur

- Les contrôles Web sont des outils plus complets en fonctionnalités que les contrôles HTML. Ils héritent d'une classe plus riche que le contrôle HTML
- La syntaxe des contrôles Web est :

```
<asp:NomControle  
id="nom_de_mon_controle"  
runat="server">  
</asp:NomControle>
```

System.Object

System.Web.UI.Control

System.Web.UI.WebControls.WebControl

System.Web.UI.ScriptControl

System.Web.UI.WebControls.BaseDataBoundControl

System.Web.UI.WebControls.BaseDataList

System.Web.UI.WebControls.Button

System.Web.UI.WebControls.Calendar

System.Web.UI.WebControls.CheckBox

System.Web.UI.WebControls.CompositeControl

System.Web.UI.WebControls.DataListItem

System.Web.UI.WebControls.FileUpload

System.Web.UI.WebControls.HyperLink

System.Web.UI.WebControls.Image

System.Web.UI.WebControls.Label

System.Web.UI.WebControls.LinkButton

System.Web.UI.WebControls.LoginName

System.Web.UI.WebControls.Panel

System.Web.UI.WebControls.SiteMapNodeItem

System.Web.UI.WebControls.Table

System.Web.UI.WebControls.TableCell

System.Web.UI.WebControls.TableRow

System.Web.UI.WebControls.TextBox

System.Web.UI.WebControls.ValidationSummary

# ASP.Net

## Principes du contrôle serveur

- Parmi les attributs les plus utiles qui existent autant pour les contrôles serveur Web que HTML on a :

Attributs	Type	Description
Visible	Boolean	Permet d'afficher ou dissimuler un contrôle
Id	String	Identifiant permettant d'appeler un objet en code behind
Parent	Control	Référence vers le contrôle parent

- Pour modifier les attributs/propriétés d'un contrôle serveur on dispose de existe trois façons : code aspx, code behind et fenêtre propriétés

# ASP.Net

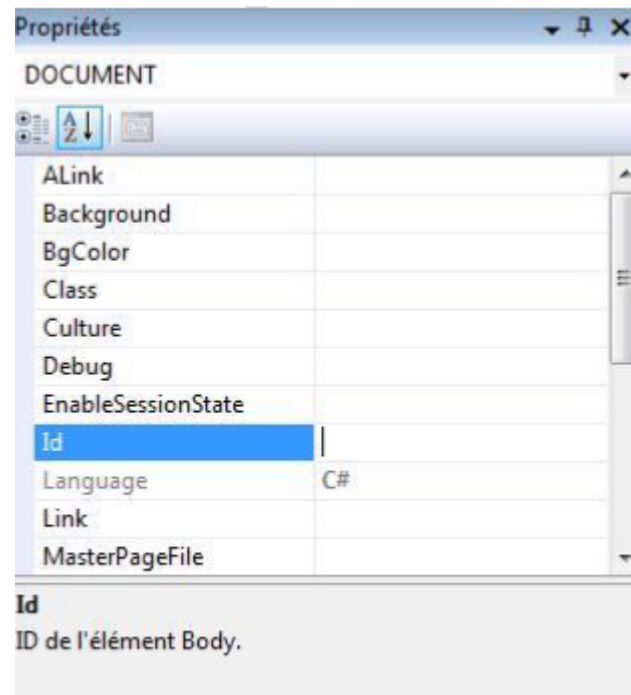
## Principes du contrôle serveur : modifier les attributs

1) A partir du code de la page ASPX

Il suffit de rajouter sur la balise à laquelle on veut changer l'attribut avec sa valeur. Ce sera de la forme « MonAttribut="MaValeur" ».

Exemple : <body id="test">

2) A l'aide de la fenêtre propriétés



# ASP.Net

## Principes du contrôle serveur : modifier les attributs

3) A partir du code behind

- Avant cela il faut mettre un attribut Id sur le contrôle serveur.
- Dans le code behind il faut ajouter une ligne de cette forme :

```
MonId.MonAttribut = "MaValeur";
```

- Code à mettre dans la page aspx :

```
<form runat="server">
```

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

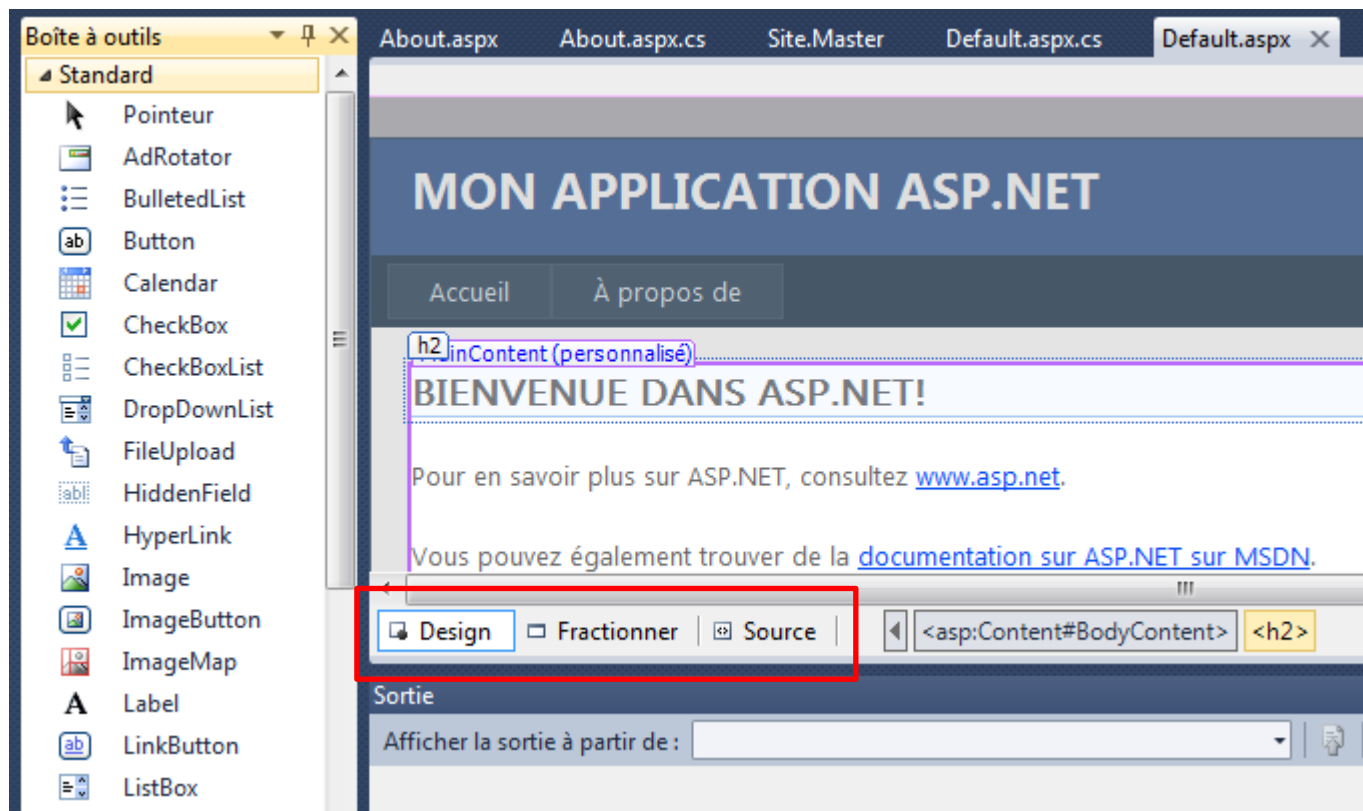
```
</form>
```

```
Button1.ID= "test";
```

# ASP.Net

## Ajouter des contrôles serveur

- Pour ajouter un contrôle serveur (qu'il soit Web ou HTML) il est possible de faire un drag & drop depuis la toolbox.
- Il existe 3 modes d'affichage : fractionner, design et source et on peut faire le drag & drop dans chacun d'eux



# ASP.Net

## Ajouter des contrôles serveur

- Il est aussi possible de créer un contrôle serveur depuis le code behind.  
Exp:

```
TextBox MonTextBox = new TextBox();  
MonTextBox.ID = "login";  
MonTextBox.Visible = true;  
form1.Controls.Add(MonTextBox);
```

Explication : Tout d'abord on instancie un objet nommé « MonTextBox » de type TextBox. Ensuite on lui applique un ID « login » et on le rend visible.



# ASP.Net

## Ajouter des contrôles serveur

### Label :

- Un contrôle Label est un contrôle qui permet d'afficher du texte sur une page web qui peut être modifié dynamiquement par le serveur

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

### Button :

- Un contrôle Button est un contrôle sur lequel on peut cliquer (bouton poussoir) et qui déclenchera alors lePostBack de la page (par défaut). Après lePostBack il exécutera la méthode qui lui est associé.

```
<input id="Button" type="button" value="button" />
```

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

# ASP.Net

## Ajouter des contrôles serveur

### TextBox :

- Un contrôle TextBox est un contrôle qui permet à l'utilisateur d'entrer du texte (c'est une zone de saisie).
- Cette zone de saisie peut être SingleLine, sur une seule ligne, ou MultiLine, sur plusieurs lignes. Le contrôle Web, par défaut, est SingleLine

### HTML

```
<input id="Text1" type="text" /> <%-- SingleLine --%>  
<textarea runat="server">Texte</textarea> <%-- Multiline --%>
```

### Web

```
<asp:TextBox ID="TextBox1" runat="server" TextMode="SingleLine">  
</asp:TextBox> <%-- SingleLine --%>
```

```
<asp:TextBox ID="TextBox1" runat="server" TextMode="MultiLine">  
</asp:TextBox> <%-- MultiLine --%>
```

# ASP.Net

## Ajouter des contrôles serveur

### DropDownList :

- Une DropDownList est une liste déroulante dans laquelle on ne peut sélectionner qu'un seul élément.

#### HTML

```
< select id="Select1">  
<option>Choix 1</option>  
<option>Choix 2</option>  
</select>
```

#### Web

```
<asp:DropDownList ID="DropDownList1" runat="server">  
<asp:ListItem Text="Choix 1" />  
<asp:ListItem Text="Choix 2" />  
</asp:DropDownList>
```

# ASP.Net

## Ajouter des contrôles serveur

### CheckBox :

- Les contrôles CheckBox sont des cases à cocher qui peuvent être assimilées à un booléen dans le sens où il permet de choisir entre deux état : sélectionné ou non (true ou false).

### HTML

```
<input id="Checkbox2" type="checkbox" />
```

### Web

```
<asp:CheckBox ID="CheckBox1" runat="server"/> <%-- CheckBox simple --%>
```

# ASP.Net

## Ajouter des contrôles serveur

### RadioButton :

- Les RadioButton sont un contrôle serveur garantissant l'unicité de la sélection dans une liste de propositions.

### HTML

```
<input id="Radio1" type="radio" value="maValeur" />
```

### Web

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server">  
<asp:ListItem Value="maValeur" Text="texte_a_afficher" Selected="True" />  
<asp:ListItem Value="maValeur2" Text="texte_a_afficher2" Selected="False" />  
</asp:RadioButtonList>
```

# ASP.Net

## Ajouter des contrôles serveur

### RadioButton :

- Les RadioButtonList permettent de grouper les RadioButton qui les composent.
  - ListItem correspond à un RadioButton.
  - Value définit la valeur de la RadioButton (par exemple « oui » ou « non »).
  - Text sera le texte associé au RadioButton.
  - Selected définit si le RadioBouton est coché ou non par défaut (en HTML il faut mettre checked="checked" pour dire qu'il est coché).

# ASP.Net

## Les événements

- Un événement est une action qui va pointer vers une méthode dans le code behind.
- Cela permet de programmer ce qui se passe lors d'un événement.
- Lorsque l'on définit un événement sur un contrôle on parle d'abonner le contrôle à un événement.
- Il existe deux méthodes pour abonner un contrôle à un événement : la méthode manuelle et la méthode graphique.

# ASP.Net

## Les événements : Exemple

- Code à placer dans la page aspx:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

```
<asp:Button ID="Button1" runat="server" Text="Cliquer ici"  
onclick="Methode_Click" />
```

```
<asp:Label ID="Label1" runat="server" Text="Vide"></asp:Label>
```

- Code à placer dans le code behind :

```
protected void Methode_Click(object sender, EventArgs e)  
{  
    Label1.Text = TextBox1.Text;  
}
```



# ASP.Net

## Cycle de vie de la page

- il y a plusieurs évènements durant la vie de la page. La page effectue un cycle durant lequel elle initialise ses contrôles enfants
- Durant le cycle de vie de la page, suivant le moment où l'on fait les actions il se peut que les objets ne soient pas encore instanciés ou qu'ils ne soient plus modifiables
- Les phases de ce cycle sont :
  - Initialisation
  - Chargement
  - Rendu
  - fermeture

# ASP.Net

## Cycle de vie de la page

- Phase d'initialisation :

1. PreInit :

- Avant tout calcul de la page (rien n'est initialisé).
- Permet de changer le thème (seul moment où l'on peut changer le css).
- Déterminer si la page est chargée pour la première fois (propriété IsPostBack du contrôle Page).

2. Init :

- Apparence chargée
- Contrôles initialisés
- Permet de lire / initialiser des propriétés de contrôle.

3. InitComplete

- Fin de la phase d'initialisation (tous les objets sont initialisés)

# ASP.Net

## Cycle de vie de la page

- Phase de chargement :

1. Preload :

- A utiliser si on a besoin d'effectuer une action relative au chargement avant l'évènement Load.

2. Load :

- Charge récursivement les contrôles enfants de la page

3. Load Complete :

- Pour les actions nécessitant que l'ensemble de la page soit chargée

# ASP.Net

## Cycle de vie de la page

- Phase de rendu :

1. Prerender :
  - Dernier évènement qui permet de modifier le contenu de la page et des contrôles
  - Permet de remplir les contrôles avec des données s'il y a lieu (par rapport à une source de données telle qu'une base SQL, un fichier XML ...)
2. SaveStateControl :
  - A partir de cet évènement, toute modification de la page et/ou de ses contrôles sera ignorée.
  - Permet de sauvegarder l'état des contrôles tels qu'ils seront affichés.
3. Render :
  - Génération du code HTML pour le navigateur.
  - Render n'est pas un évènement (c'est une méthode).

# ASP.Net

## Cycle de vie de la page

- Phase de fermeture :

1. Unload :

- Utilisé pour fermer les connexions à toutes les sources de données et/ou vider les variables temporaires en fin vie de la page.

# ASP.Net

## PostBack d'une page

- Le post-back est le modèle utilisé en ASP.Net pour transmettre les données d'un formulaire HTML avec la méthode POST.
- Ce modèle consiste à afficher et à traiter les données recueillies du formulaire sur la même page aspx
- La propriété **Page.IsPostBack** obtient une valeur qui indique si la page est en cours de rendu pour la première fois ou si elle est en cours de chargement en réponse à une publication (postback).

```
private void Page_Load()  
{  
    if (!IsPostBack)  
    {  
        Validate();  
    }  
}
```

# ASP.Net

## Script de traitements

- Tout script écrit en C# dans la page .aspx pour ASP.Net doit se trouver entre les balises suivantes

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

Exemple :

Fonction de date

```
<body>
```

```
    <form runat="server ">
```

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

```
    </form>
```

```
</body>
```

# ASP.Net

## Script de traitements

Exemple (suite):

```
<head runat="server">
  <title></title>
  <script language="C#"
    runat="server">
    string Demain()
    {
      DateTime Jour = DateTime.Now;
      Jour = Jour.AddDays(1);
      return Jour.ToString();
    }
  </script>
</head>
```

```
<body>
  <form id="form1" runat="server">
    <div>
      Date du jour :
      <% =DateTime.Now.ToString() %><br>
      Date de demain :
      <% =Demain() %>
    </div>
  </form>
</body>
```



# ASP.Net

## **fichier Global.asax**

- La classe Global est la classe capable de gérer des événements du niveau application.
- Elle se trouve dans un fichier appelé Global.asax (plus précisément Global.asax.cs).
- Le Global.asax est optionnel mais il peut faciliter le développement ainsi que la maintenance d'applications.
  - On l'utilisera, par exemple , pour écrire une entrée dans un fichier de logs lorsqu'une exception est lancée et n'est pas gérée. Il existe également d'autres utilisations possibles comme le calcul du temps nécessaire à une requête, un compteur de hits, ...
- Concrètement, la classe Global est une classe dont il n'existe qu'une instance (singleton)
- Ce fichier se trouve obligatoirement dans le répertoire racine de l'application ASP.Net. Il y en a un et un seul par application.

# ASP.Net

## **fichier Global.asax**

- Accès a une variable statique du fichier

```
public class Global : System.Web.HttpApplication
{
    public static readonly string ConnectionString="connection infos" ;
    public static int numeroBidon;
// Méthodes
}
string strConn = Global.ConnectionString ;
//ajouter cette ligne dans un formulaire
```

# ASP.Net

## Gestion d'états

- Lors de l'affichage d'une page Web, il est possible de personnaliser cet affichage en fonction de l'utilisateur qui s'y connecte.
- Ainsi, on peut conserver des mots de passe pour éviter des identifications trop fréquentes qui peuvent être lourdes à la longue.
- Il est également possible d'afficher les informations personnelles d'un utilisateur sur sa page d'accueil ou encore personnaliser le style du site selon son désir.
- Dans cette optique, deux moyens distincts existent:
  - La gestion des états côté client
  - La gestion des états côté serveur.

# ASP.Net

## **Gestion d'états**

Avantages et inconvénients de ces deux alternatives

1-Côté Client :

### **Performance :**

Les requêtes s'effectuant directement sur le navigateur des clients, le serveur est moins surchargé et gagne en vitesse d'affichage.

### **Pérennité des informations :**

Un stockage côté client des informations par cookies permet un accès plus rapide dans un site nécessitant une authentification.

L'utilisateur n'a pas besoin de se réauthentifier tant que les cookies n'ont pas été effacés ou ne sont pas périmés.

# ASP.Net

## **Gestion d'états**

Avantages et inconvénients de ces deux alternatives

2- Côté Serveur :

### **Sécurité :**

Du fait que les informations ne sont pas stockées sur un poste lambda, elles ont moins de chances d'être perdues, par ailleurs il n'est pas possible de modifier ces informations en cours de navigation car elles transitent de façon transparente pour le client, ce qui n'est pas le cas des QueryString ou des Cookies.

### **Réduction de la bande Passante :**

Si l'on doit stocker de lourdes informations, ce stockage côté client peut entraîner une augmentation de la bande passante du fait de ce perpétuel vas-et-viens entre le client et le serveur. Il est alors plus judicieux de stocker ces informations côté serveur qui sera un meilleur compromis

# ASP.Net

## Gestion d'états

1- La gestion d'états côté client :

Voici une liste des techniques utilisées pour ce mode de gestion

**View State** : On utilise les View State pour traquer les valeurs dans des contrôles. Il est possible d'ajouter des valeurs personnalisées à nos View State.

**Hidden Field**: Il est parfois utile de créer des champs cachés (Hidden Field) dans lesquels seront stockés des valeurs que l'on pourra utiliser après unPostBack ou un Cross Posting.

**Cookies** : Les Cookies sont un bon moyen pour récupérer des données sur toutes les pages d'un même site Web. Les valeurs sont stockées du côté client par le navigateur Web.

**Query Strings**: On peut transmettre des données d'une page à l'autre en les faisant passer directement dans l'URL, on appelle cela le QueryString

# ASP.Net

## Gestion d'état

### Gestion d'états

#### 2- La gestion d'états côté serveur:

Il existe deux moyens en ASP.Net pour pouvoir passer des informations entre les pages du côté serveur :

#### **Application State**

permet de stocker des informations qui seront accessibles dans toutes les pages Web de l'application, elles sont accessible par tout le monde

#### **Session**

étant spécifique à un utilisateur (avec un identifiant unique), seule la personne ayant le bon identifiant pourra accéder aux données

# ASP.Net

## Passage d'arguments via url

- Pour rediriger l'utilisateur vers une page donnée avec des paramètres on utilise:

```
Response.Redirect("MesInfos.aspx?L=" + TbLogin.Text + "&P=" +  
TbPasswd.Text);
```

- Pour récupérer les valeurs des paramètres passés dans l'URL

```
if (Request.Params["L"]!=null)
```

```
La.Text += "<br>Login: " + Request.Params["L"].ToString();
```

Ou bien

```
La.Text += "<br>Login: " + Request.QueryString["fullname"]. ToString();
```



# ASP.Net

## Les variables de session

- Les variables de session sont stockées en mémoire sur le serveur.
- Il peut y avoir autant de variables de session que l'on le désire, cependant il faut faire attention au performance
- 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.

```
Session["FirstName"] = FirstNameTextBox.Text;  
Session["LastName"] = LastNameTextBox.Text;  
if (Session["Username"]!=null)  
La.Text += "<br>Login: " + Session["Username"].ToString();
```

# ASP.Net

## la mise en cache

- Le cache de données est un contexte de données, situé côté serveur, qui est partagé entre tous les utilisateurs.
  - Autrement dit, si un utilisateur ajoute ou modifie une donnée dans le contexte de cache, alors tout autre utilisateur peut lire cette donnée, et même la modifier.
- La mise en cache consiste donc a stocker des données fréquemment consultées ou des pages Web entières dans la mémoire, où ils peuvent être récupérés plus rapidement que d'un fichier ou base de données
- L'objectif est d'améliorer les performances.

• Exp :

```
Cache["Greeting"] = "Hello, cache!";  
if (Cache["Greeting"] != null)  
    Label1.Text = (string)Cache["Greeting"];  
else  
    Label1.Text = "Hello, world!";
```

# ASP.Net

## **fichier de configuration Web.config**

- ASP.Net permet de spécifier des paramètres de configuration qui affectent :
  - toutes les applications Web sur un serveur
  - seulement une application unique
  - des pages individuelles
  - dossiers individuels
- On peut créer des paramètres de configuration pour des fonctionnalités telles que les options du compilateur, le débogage, l'authentification utilisateur, l'affichage des messages d'erreur, les chaînes de connexion, etc.
- Les données de configuration sont stockées dans des fichiers XML nommés Web.config.

# ASP.Net

## **fichier de configuration Web.config**

Exemple : Erreurs personnalisées

On peut configurer la façon dont les informations sont affichées par ASP.Net lorsque des erreurs non gérées se produisent pendant l'exécution d'une requête Web

```
<customErrors mode="RemoteOnly"
  defaultRedirect="GenericErrorPage.htm">
  <error statusCode="403" redirect="NoAccess.htm" />
  <error statusCode="404" redirect="FileNotFound.htm" />
</customErrors>
```

# ASP.Net

## Validation des données de l'utilisateur

- Un élément essentiel que le développeur doit gérer : S'assurer que les données saisis par l'utilisateur sont valides
- Les contrôles de validation se trouvent dans la boîte à outils de Visual Studio
- Pour mettre en œuvre une validation on associe au contrôle à valider le contrôle de validation adéquat
- Lors de l'utilisation des contrôles de validation ASP.Net génère un code en JavaScript qui s'exécutera lorsque l'utilisateur quitte le focus du contrôle à valider
- La validation côté client est activée par défaut, pour la désactiver il faut mettre la valeur de la propriété `EnableClientScript` à `False` dans le contrôle de validation

# ASP.Net

- Voici les étapes à suivre pour mettre en place une validation :
  1. Ouvrir une page aspx puis ajouter le contrôle web qui devra être validé (Textbox, liste déroulante, ...)
  2. Aller dans le bloc des contrôles de validation de la boîte à outils et glisser le contrôle de validation qu'on veut utiliser à côté du contrôle à valider
  3. Donner une valeur à la propriété ID du contrôle de validation (donner une valeur significative)
  4. Attribuer à la propriété ControlToValidate du contrôle de validation l'ID du contrôle à valider
  5. Attribuer à la propriété ErrorMessage du contrôle de validation un message d'erreur significatif pour l'utilisateur

# ASP.Net

6. Attribuer à la propriété Text du contrôle de validation un message qui informe l'utilisateur qu'il y a une erreur (asterisk (\*)) Par exemple (dans ce cas la valeur de ErrorMessage ne sera pas visible à côté du contrôle quand la validation échoue)
  7. Assigner une valeur à la propriété ToolTip du contrôle de validation
  8. Attribuer à la propriété Display du contrôle de validation une valeur:
    - None: Le message d'erreur ne s'affiche pas
    - Static : Le message d'erreur occupe un espace blanc
    - Dynamic : Dans ce cas il n'y a pas d'espace blanc
  9. On peut ajouter également le contrôle ValidationSummary pour afficher tous les messages d'erreur dans un unique endroit de la page web, cela est utile lorsque la page contient plusieurs contrôles à valider et qu'il est difficile de gérer l'affichage des messages
- Lorsque la propriété ShowMessageBox du ValidationSummary est vraie, les messages d'erreurs s'afficheront dans une pop-up

# ASP.Net

- Lorsque l'utilisateur quitte le focus d'un contrôle la validation coté client se déclenche
- Dans certains cas cela pose un problème : Si un utilisateur clique sur un bouton Annuler dans un formulaire qui contient des données non valide alors dans ce cas la validation coté client se déclenche par conséquent il est impossible d annuler la saisie
- Pour résoudre ce problème mettre la propriété CausesValidation a False dans les contrôles qui ne doivent pas utiliser la validation (bouton Annuler, Initialiser, ...)



# ASP.Net

Voici les principaux contrôles de validation:

- **RequiredFieldValidator** : pour s'assurer que l'utilisateur a saisi une valeur (différente de la chaîne vide)
- **CompareValidator** : compare l'entrée d'utilisateur à la valeur d'une constante, à la valeur d'un autre contrôle (à l'aide d'un opérateur de comparaison tel que « inférieur à, égal ou supérieur à ») ou à un type de données spécifique.
- **RangeValidator** : vérifie si les valeurs d'un autre contrôle sont comprises dans une plage acceptable, où les valeurs minimales et maximales sont fournies soit directement, soit par référence à un autre contrôle
- **RegularExpressionValidator** : Vérifie que l'entrée correspond à un critère défini par une expression régulière (adresses électroniques, numéros de téléphone, codes postaux, ...)
- **CustomValidator** : Vérifie une entrée d'utilisateur à l'aide d'une logique de validation (par code). Ce type de validation permet de vérifier les valeurs dérivées au moment de l'exécution. La validation peut être faite côté serveur ou côté client

# IIS Express




- IIS Express est le serveur Web par défaut pour les projets d'application Web dans Visual Studio (on peut choisir un autre)
- IIS Express est conçu pour émuler IIS (il est recommandé de faire un test avant de passer en prod)
- Il est recommandé pour un environnement de Dev

# IIS Express

IIS Express offre les fonctionnalités suivantes:

- Il prend en charge et active le même modèle d'extensibilité et les paramètres du fichier Web.config que IIS
- Il ne nécessite pas de modifications dans le code de l'application Web.
- Il peut être installé côte à côte avec la version complète d'IIS et d'autres serveurs Web.
  - On peut choisir un serveur Web différent pour chaque projet.

# IIS Express

- IIS Express démarre automatiquement lorsqu'on exécute un projet dans Visual Studio et il s'arrête lorsqu'on click sur  ou on ferme le projet.
- Lorsque IIS Express est lancé, il affiche une icône dans la barre d'état système

