Gestion d'état

Sommaire

1 Introduction

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 passes 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 s'offrent à nous :

- La gestion des états côté client ;
- La gestion des états côté serveur.

Dans la partie cliente, il est possible de conserver des informations de l'utilisateur dans l'URL ou dans des Cookies. La partie serveur, quant à elle, peut récupérer des informations à partir des Cookies ou de l'URL mais présente l'avantage de pouvoir stocker ou encore requêter sur une base de donnée (MySQL, Oracle, Acess, XML, etc…).

Dans ce chapitre nous étudierons ces deux méthodes en détail et ce qui vous permettra de choisir dans quel cas utiliser plutôt l'une que l'autre.

Avant de nous lancer, faisons un point sur les avantages et les inconvénients de ces deux alternatives.

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 ré-authentifier tant que les *cookies* n'ont pas été effacés ou ne sont pas périmés.

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.

2 La gestion d'état côté client

La façon la plus performante pour la navigation est la sauvegarde d'information côté client. Pour ce faire, nous avons à notre disposition divers outils que nous expliquerons au cours de cette partie.

Voici une liste des outils que nous allons vous présenter :

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.*
Control Sate	Quand on crée un *Custom Control* qui a besoin d'un *View State* pour profiter de la fonctionnalité voulu, il est possible de créer un *Control State* pour permettre aux autres développeurs de ne pas « casser » votre contrôle en désactivant le *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 un PostBack 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 coté 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*.

2.1 Le View State

Vous avez surement du remarquer que lorsque vous cliquez sur un *button* d'une page ASPX, les informations que vous avez rentré auparavant sont automatiquement conservées (sauf pour les éléments rajoutés en *CodeBehind*).

Par exemple si on met à jour un *label*, tous les rechargements de la page qui suivront conserveront cette dernière mise à jour. Ceci est possible grâce à la gestion d'état côté client d'ASP.NET et plus particulièrement aux *View State.*

La propriété *View State* nous fournit une collection d'objet permettant de conserver des valeurs entre plusieurs requêtes sur la même page. Quand une page ASP.NET est traitée, l'état courant de la page ainsi que de ses contrôles sont hachés en une chaîne de caractère puis sauvegardés dans la page un peu comme un champ caché en HTML (`<input type="hidden" />`).

Si les données sont trop longues (que l'on peut spécifier grâce à la propriété *MaxPageStateField-Length*) alors ASP.NET scindera nos données en de multiples champs cachés plutôt qu'en un seul.

Dans les sous-parties suivantes nous aborderons le chiffrement des données *View State*, la désactivation des *View State* ainsi que l'ajout de données personnalisées dans le *View State*.

2.1.1 Chiffrement des données *View State*

Il est possible d'activer le chiffrement des données *View State* afin de rendre plus difficile l'accès des données aux personnes malintentionnées. Cette mise en place créera une charge

supplémentaire de calcul pour le serveur mais peut être nécessaire si vous décidez de stocker des informations personnelles dans les *View States*.

Pour ce faire, il faut se rendre dans votre fichier de configuration *Web.config* (voir chapitre précédent pour la configuration en détail de ce fichier). Il faut alors modifier l'attribut de *pages viewStateEncryptionMode* en *Always* comme le montre l'exemple ci-dessous :

```
<configuration>
  <system.web>
    <pages viewStateEncryptionMode="Always"/>
  </system.web>
</configuration>
```

Vous pouvez sinon activer le chiffrement de *View State* pour des pages spécifiques en ajoutant la propriété `ViewStateEncryptionMode="Always"` dans la directive page de la page Web voulue comme le montre cet exemple :

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Page1.aspx.cs"
Inherits="Test.Page1" ViewStateEncryptionMode="Always"%>
```

2.1.2 Désactiver le *View State*

Le *View State* est activé par défaut pour chaque contrôle. Cela dit, si vous ne l'utilisez pas, il prend une charge de calcul inutile. Vous pouvez palier à ce problème en mettant la propriété `EnableViewState` à `false` pour chaque contrôle.

Exemple :

```
<asp:Label ID="RecupData" runat="server" Text="Label"
EnableViewState="false"></asp:Label>
```

Il est aussi possible de le désactiver dans le *Web.config* (avec la balise pages) ou dans la directive Page d'une page spécifique avec la même propriété (EnableViewState).

2.1.3 Lire et écrire des données personnalisés dans le *View State*

Il est possible d'ajouter ou de récupérer des valeurs personnalisées dans des *View States*. Par exemple, vous souhaitez conserver une valeur particulière pendant toute la durée de la visite de l'utilisateur, alors l'ajout d'une donnée personnalisée dans le *View State* sera le moyen le plus efficace et le plus sûr pour y parvenir. En contrepartie cette valeur ne sera accessible que tant que l'utilisateur ne visitera pas d'autre page. Ceci est une bonne alternative pour le stockage de données temporaires.

Exemple :

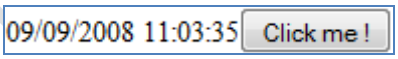
C#

```
protected void Page_Load(object sender, EventArgs e)
{
    // On crée nos contrôles dynamiquement
    //on peut aussi utiliser ceux déjà crée dans notre page Web
    Label Label1 = new Label();

    Button Button1 = new Button();
    Button1.Text = "Click me !";

    // On les ajoute à notre page Web
    form1.Controls.Add(Label1);
    form1.Controls.Add(Button1);

    // Si notre ViewState n'est pas null
    if (ViewState["LastAction"] != null)
    {
       // On affiche ce qui est contenu dans notre ViewState
       Label1.Text = (String)ViewState["LastAction"];
       // On peut remarquer que le mot clé String permet de
       //confirmer que nous envoyons une chaîne de caractère
       //à notre Label.

       // Un autre type d'objet aurait pu être passé en paramêtre
       //il aurait cependant fallu ajouter la méthode .ToString()
       //comme ceci :
       //  ViewState.Add("LastAction", DateTime.Now);
       //  Label1.Text = ((DateTime)ViewState["LastAction"]).ToString();
    }
    else
    {
       Label1.Text = "Pas d'action précédente";
    }
    // On crée notre ViewState
    ViewState.Add("LastAction", DateTime.Now.ToString());
    // Bien que ce ne soit pas obligatoire comme dit plus haut
    //on ajoute directement à notre ViewState la méthode ToString()           }
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    ' On crée nos contrôles dynamiquement
    ' on peut aussi utiliser ceux déjà crée dans notre page Web
    Dim Label1 As Label = New Label()

    Dim Button1 As Button = New Button()
    Button1.Text = "Click me !"

    ' On les ajoute à notre page Web
    form1.Controls.Add(Label1)
    form1.Controls.Add(Button1)

    ' Si notre ViewState n'est pas null
    If ViewState("LastAction") <> Nothing Then
        ' On affiche ce qui est contenu dans notre ViewState
        Label1.Text = CType(ViewState("LastAction"), String)
        ' On peut remarquer que le mot clé String permet de
        ' confirmer que nous envoyons une chaîne de caractère
        ' à notre Label.

        ' Un autre type d'objet aurait pu être passé en paramètre
        ' il aurait cependant fallu ajouter la méthode .ToString()
        ' comme ceci :
        ' ViewState.Add("LastAction", DateTime.Now)
        ' Label1.Text = CType(ViewState("LastAction").ToString(), DateTime)
    Else
        Label1.Text = "Pas d'action précédente"

        ' On crée notre ViewState
        ViewState.Add("LastAction", DateTime.Now.ToString())
        ' Bien que ce ne soit pas obligatoire comme dit plus haut
        ' on ajoute directement à notre ViewState la méthode ToString()          }
    End If
End Sub
```

Une fois avoir cliqué sur le bouton voici ce à quoi votre page doit ressembler :



2.2 Le Control State

Lorsque vous créez un *Custom Control* utilisant les *View States*, le développeur qui utilisera votre contrôle pourra désactiver ces *View States*, ce qui aura pour effet d'empêcher votre contrôle de fonctionner correctement. Pour résoudre ce problème, vous pouvez utiliser la propriété *Control State* afin de stocker les informations relatives à votre contrôle. Le *Control State* est une propriété qui vous permet de conserver les informations qui sont spécifiques à votre contrôle et qui ne peut pas être désactivée à l'inverse du *View State*.

Pour utiliser le *Control State* dans un *Custom Control* vous devez surcharger la méthode *OnInit()* et appeler la méthode *RegisterRequiresControlState* pendant l'initialisation, puis surcharger les méthodes *SaveControlState* et *LoadControlState*.

Exemple :

C#

```
public class Class1 : Label
    {
        protected override void Render(HtmlTextWriter writer)
        {
            base.Render(writer);
            writer.Write("Test de ma classe héritant de Label");
        }

        protected override void OnInit(EventArgs e)
        {
            base.OnInit(e);
            Page.RegisterRequiresControlState(this);
        }

        protected override object SaveControlState()
        {
            return base.SaveControlState();
        }

        protected override void LoadControlState(object savedState)
        {
            base.LoadControlState(savedState);
        }
    }
```

VB.NET

```
Public Class Class1
    Inherits Label

    Protected Overrides Sub Render(ByVal writer As HtmlTextWriter)
        MyBase.Render(writer)
        writer.Write("Test de ma classe héritant de Label")
    End Sub

    Protected Overrides Sub OnInit(ByVal e As EventArgs)
        MyBase.OnInit(e)
        Page.RegisterRequiresControlState(Me)
    End Sub

    Protected Overrides Function SaveControlState() As Object
        Return MyBase.SaveControlState()
    End Function

    Protected Overrides Sub LoadControlState(ByVal savedState As Object)
        MyBase.LoadControlState(savedState)
    End Sub


End Class
```

Il faut garder à l'esprit qu'un *Custom Control* doit rester le plus générique possible. Ne pas donner la possibilité au développeur de choisir s'il souhaite conserver les *View States* ou non peu faire perdre tout l'intérêt de votre contrôle. Cependant nous saurons qu'il est possible de le faire si besoin. Et voici ce que l'on obtient : Test de ma classe héritant de Label

2.3 Les Hidden Fields

Les *View State* stockent les informations sur la page Web en utilisant des *HiddenField*. Les *HiddenField* sont renvoyés sur le serveur quand l'utilisateur valide un formulaire. Cependant les informations ne sont jamais affichées par le navigateur (sauf si l'utilisateur décide d'afficher le code source de la page). ASP.NET nous donne la possibilité de créer nous même ces *HiddenField* et de stocker nos propres valeurs à l'intérieur de celles-ci.

Le contrôle *HiddenField* ne peut stocker qu'une seule variable dans sa propriété *Value* et doit être ajouté via le code source dans la page. L'utilisation manuelle des *HiddenField*, à la différence des *View States*, n'ont ni compression, ni chiffrement ou hachage, un utilisateur avancé pourra consulter voire modifier les informations contenus dans nos *HiddenField*.

Exemple d'utilisation :

ASPX de Page1.aspx

```
<body>
    <form id="form1" runat="server">
    <br />
    <%-- Création de notre HiddenField --%>
    <asp:HiddenField ID="HiddenField1" runat="server" Value="123456" />
    <%-- Le boutton ci-dessous nous permettra d'envoyer notre HiddenField par Cross
Posting --%>
    <asp:Button ID="Button1" runat="server" Text="Envoyer"
PostBackUrl="~/Page2.aspx" />
    </form>
</body>
```

ASPX de Page2.aspx

```
<body>
    <form id="form1" runat="server">
    <div>
        <%-- Notre page contiendra un label pour l'affichage de nos données
         récupérées de la page précédente --%>
        <asp:Label ID="Label1" runat="server"></asp:Label>
        <br />
    </div>
    </form>
</body>
```

C# de Page2.aspx

```
protected void Page_Load(object sender, EventArgs e)
{
  // On utilise la propriété PreviousPage pour vérifier qu'on vient d'un page
  //qui nous a envoyé des données par Cross Posting
  if (Page.PreviousPage != null)
  {
      // Nous affichons nos informations contenus dans notre HiddenBack de la
      //page précédente
      Label1.Text = ((HiddenField)PreviousPage.FindControl("HiddenField1")).Value;
  }
}
```

VB.NET de Page2.aspx

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    ' On utilise la propriété PreviousPage pour vérifier qu'on vient d'un page
    ' qui nous a envoyé des données par Cross Posting


    If Not Page.PreviousPage Is Nothing Then
        ' Nous affichons nos informations contenus dans notre HiddenBack de la
        ' page précédente
        Label1.Text = CType(PreviousPage.FindControl("HiddenField1"),
HiddenField).Value
    End If

End Sub
```

2.4 Les Cookies

Les applications Web peuvent stocker des données de faible taille dans le navigateur client en utilisant les *Cookies*. Un *Cookie* est une petite quantité de données stockée soit dans un fichier texte dans le système de fichier client (si le *Cookie* est persistant) soit en mémoire dans les sessions du navigateur client (si le *Cookie* est temporaire). Le plus souvent, on utilise les *Cookies* pour identifier un utilisateur sur les différentes pages d'un même site Web. Ils permettent aussi d'afficher les informations relatives à cet utilisateur si on le désire.

Les *Cookies* sont le moyen le plus flexible et le plus fiable pour stocker des informations sur le poste client. Cependant, l'utilisateur peut supprimer ces *Cookies* à n'importe quel moment s'il le désire. Pour pallier à ce genre de problème il faut tout le temps tester la validité des *Cookies* et proposer une nouvelle authentification si nécessaire qui recréera un nouveau *Cookie* contenant les informations dont le site a besoin.

2.4.1 Lire et écrire des *Cookies*

Une application Web crée un *Cookie* en l'envoyant sur le client en tant qu'entête dans une réponse HTTP. Le navigateur Web renvoie ensuite ce même *Cookie* sur le serveur pour chaque nouvelle requête.

Dans l'exemple suivant nous allons voir comment créer et supprimer des Cookies en utilisant un système de login.

C# de Page1.aspx.cs

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

{
    // Si notre cookies n'est pas défini
    if (Request.Cookies["Name"] == null)
    {
        // On crée :

        // - Un label
        Label EntreNom = new Label();
        EntreNom.Text = "Entrer nom";

        // - Une TextBox
        TextBox name = new TextBox();
        name.ID = "name";

        // - Un Button
        Button Valid = new Button();
        Valid.Text = "Valider";
        //auquel on ajoute un évenement Click pointant sur SetCookie
         Valid.Click += new System.EventHandler(this.SetCookie);

        // On ajoute tout nos éléments à notre formulaire
        form1.Controls.Add(EntreNom);
        form1.Controls.Add(name);
        form1.Controls.Add(Valid);
    }
    //Sinon
    else
    {
        // On crée :
        // - Un label
        Label AfficheNom = new Label();
        // Ce label va prendre le Cookies["name"] dans sa propriété
        //texte pour en afficher le contenu
        AfficheNom.Text = (Request.Cookies["Name"].Value).ToString();

        // - Un button
        Button Deconnect = new Button();
        Deconnect.Text = "Se déconnecter";
        // On ajoute un évènement Click pointant sur UnsetCookie
        Deconnect.Click += new System.EventHandler(this.UnsetCookie);

        // On ajoute nos élément à notre formulaire
        form1.Controls.Add(AfficheNom);
        form1.Controls.Add(Deconnect);
    }
}
```

C# de Page1.aspx.cs (suite)

```
// Cette méthode va nous permettre de créer notre Cookie
protected void SetCookie(object sender, EventArgs e)
{
    // Tout d'abord on défini sa valeur que l'on récupère de notre formulaire
    Response.Cookies["Name"].Value = ((TextBox)Page.FindControl("name")).Text;

    // On crée un date d'expiration, ici dans une minute
    //si on poursuit notre navigation au delà de ce temps
    //les éléments de nos pages qui auront besoin de notre cookie
    //ne fonctionneront plus
    Response.Cookies["Name"].Expires = DateTime.Now.AddMinutes(1);
    // Il ne faut pas oublier de gérer dans nos pages Web une expiration de
    //cookie pour ne pas générer d'erreur, par exemple dans le chargement
    //d'une page, on crée une redirection automatiquevers la page
    //d'authentification si le cookie n'existe pas ou plus

    // Pour terminer on actualise la page afin qu'elle prenne en compte le cookie
    //crée
    Response.Redirect("Page1.aspx");
}

 // Cette méthode nous permet de supprimer notre Cookie
 protected void UnsetCookie(object sender, EventArgs e)
 {
    // Pour supprimer un cookie il suffit d'actualiser
    // sa date d'expiration au jour précédent par exemple
    Response.Cookies["Name"].Expires = DateTime.Now.AddDays(-1);

    // On termine par une actualisation de la page
    Response.Redirect("Page1.aspx");
 }
```

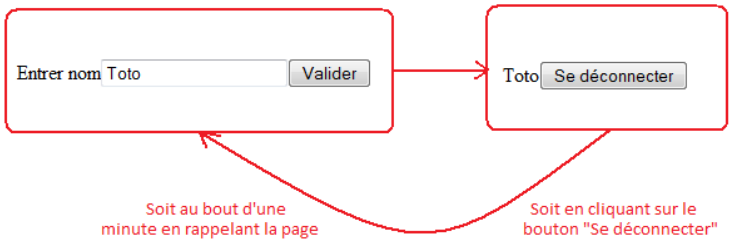
VB.NET de Page1.aspx

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    ' Si notre cookies n'est pas défini
    If Request.Cookies("Name") Is Nothing Then
        ' On crée :

        ' - Un label
        Dim EntreNom As Label = New Label()
        EntreNom.Text = "Entrer nom"

        ' - Une TextBox
        Dim name As TextBox = New TextBox()
        name.ID = "name"

        ' - Un Button
        Dim Valid As Button = New Button()
        Valid.Text = "Valider"
        ' auquel on ajoute un évenement Click pointant sur SetCookie
        AddHandler Valid.Click, AddressOf Me.SetCookie

        ' On ajoute tout nos éléments à notre formulaire
        form1.Controls.Add(EntreNom)
        form1.Controls.Add(name)
        form1.Controls.Add(Valid)
```

VB.NET de Page1.aspx (suite)

```
        'Sinon
    Else

        ' On crée :
        ' - Un label
        Dim AfficheNom As Label = New Label()
        ' Ce label va prendre le Cookies["name"] dans sa propriété
        ' texte pour en afficher le contenu
        AfficheNom.Text = (Request.Cookies("Name").Value).ToString()

        ' - Un button
        Dim Deconnect As Button = New Button()
        Deconnect.Text = "Se déconnecter"
        ' On ajoute un évènement Click pointant sur UnsetCookie
        AddHandler Deconnect.Click, AddressOf Me.UnsetCookie

        ' On ajoute nos élément à notre formulaire
        form1.Controls.Add(AfficheNom)
        form1.Controls.Add(Deconnect)
    End If
End Sub

' Cette méthode va nous permettre de créer notre Cookie
Protected Sub SetCookie(ByVal sender As Object, ByVal e As EventArgs)
    ' Tout d'abord on défini sa valeur que l'on récupère de notre formulaire
    Response.Cookies("Name").Value = CType(Page.FindControl("name"), TextBox).Text

    ' On crée un date d'expiration, ici dans une minute
    ' si on poursuit notre navigation au delà de ce temps
    ' les éléments de nos pages qui auront besoin de notre cookie
    ' ne fonctionneront plus
    Response.Cookies("Name").Expires = DateTime.Now.AddMinutes(1)
    ' Il ne faut pas oublier de gérer dans nos pages Web une expiration de
    ' cookie pour ne pas générer d'erreur, par exemple dans le chargement
    ' d'une page, on crée une redirection automatiquevers la page
    ' d'authentification si le cookie n'existe pas ou plus

    ' Pour terminer on actualise la page afin qu'elle prenne en compte le cookie
    ' crée
    Response.Redirect("Page1.aspx")
End Sub

' Cette méthode nous permet de supprimer notre Cookie
Protected Sub UnsetCookie(ByVal sender As Object, ByVal e As EventArgs)
    ' Pour supprimer un cookie il suffit d'actualiser
    ' sa date d'expiration au jour précédent par exemple
    Response.Cookies("Name").Expires = DateTime.Now.AddDays(-1)

    ' On termine par une actualisation de la page
    Response.Redirect("Page1.aspx")
End Sub
```

Aperçu :



2.4.2 Contrôler la portée des *Cookies*

Il peut arriver que vous stockiez des informations personnelles dans vos *Cookies* et que vous ne souhaitez pas que d'autre site Web puisse y accéder. Bien que par défaut, le navigateur empêche ce genre de manœuvre, vous pouvez contrôler la portée de vos *Cookies* en spécifiant un dossier ou un nom de domaine.

Exemple :

Pour illustrer cette action, nous allons reprendre la méthode *SetCookie* vu dans l'exemple ci-dessus.

C# de Page1.aspx

```
protected void SetCookie(object sender, EventArgs e)
{
    Response.Cookies["Name"].Value = ((TextBox)Page.FindControl("name")).Text;
    Response.Cookies["Name"].Expires = DateTime.Now.AddMinutes(1);

    // On peut choisir un dossier pour définir la portée
    //de nos Cookies de notre application Web
    Response.Cookies["Name"].Path = "/Test";

    // On peut aussi définir la portée d'un Cookie
    //à un nom de domaine comme le montre cet exemple
    Response.Cookies["Name"].Domain = "MonDomaine.com";

    Response.Redirect("Page1.aspx");
}
```

VB.NET de Page1.aspx

```
Protected Sub SetCookie(ByVal sender As Object, ByVal e As EventArgs)
    Response.Cookies("Name").Value = CType(Page.FindControl("name"), TextBox).Text
    Response.Cookies("Name").Expires = DateTime.Now.AddMinutes(1)

    ' On peut choisir un dossier pour définir la portée
    ' de nos Cookies de notre application Web
    Response.Cookies("Name").Path = "/Test"

    ' On peut aussi définir la portée d'un Cookie
    ' à un nom de domaine comme le montre cet exemple
    Response.Cookies("Name").Domain = "MonDomaine.com"

    Response.Redirect("Page1.aspx")
End Sub
```

N.B. : Concernant la propriété *Domain*, dans le cas présent on a limité la portée du *Cookie* au domaine entier (intranet.MonDomaine.com ou www.Mondomaine.com appartiennent au même domaine), il est cela dit possible de limiter la portée du Cookie à un nom d'hôte complet.

2.4.3 Stocker plusieurs valeurs dans un seul *Cookie*

Il existe une limite quant à l'utilisation des *Cookies* et bien que cela dépende des navigateurs, on peut dire qu'en règle générale on ne peut travailler qu'avec 20 *Cookies* par site, chaque *Cookies* ne pouvant pas dépasser les 4 Ko.

Pour nous permettre de mieux utiliser cette ressource nous allons voir comment stocker plusieurs valeurs dans un seul *Cookie*.

En effet, un *Cookie* peut se comporter comme un tableau à une dimension. Ainsi, un peu comme si nous donnions le nom de notre colonne et de notre ligne, nous pouvons indiquer à notre *Cookie* les différentes valeurs à prendre.

Exemple :

C#
```
Response.Cookies["user"]["Lastname"] = "TOTO";
Response.Cookies["user"]["Firstname"] = "Titi";
Response.Cookies["user"].Expires = DateTime.Now.AddDays(1);
```

VB.NET
```
Response.Cookies("user")("Lastname") = "TOTO"
Response.Cookies("user")("Firstname") = "Titi"
Response.Cookies("user").Expires = DateTime.Now.AddDays(1)
```

Comme on l'a dit précédemment, les *Cookies* peuvent contenir d'autres Objet que des String. Le mot clé *Value* est ici inutile. En revanche, les conditions d'expiration ou de portées se définissent sur le premier index (dans cet exemple c'est `user`).

2.5 Les Query Strings

Les *Query Strings* permettent de faire passer des données directement dans l'URL. Comme ces informations sont visibles facilement par l'utilisateur, il est recommandé de n'utiliser les *Query String* que dans le cas où les données peuvent être modifiable sans porter préjudice à la vie privée des utilisateurs ou bien à la viabilité du site Web.

On peut éventuellement s'en servir pour récupérer le numéro de la page que nous somme en train de consulter dans un forum.

Dans l'exemple suivant nous allons vous montrer comment créer et récupérer des *Query Strings.*

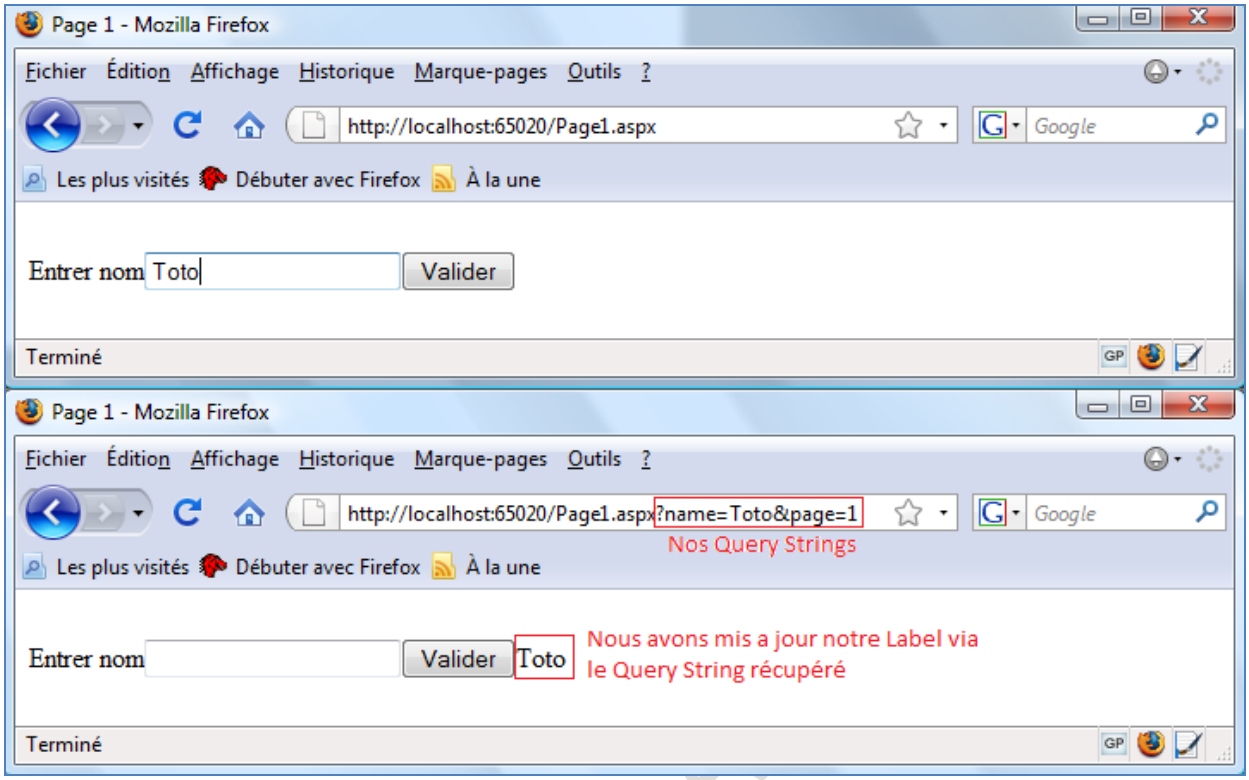
C# de Page1.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    Label EntreNom = new Label();
    EntreNom.Text = "Entrer nom";

    TextBox name = new TextBox();
    name.ID = "name";

    Button Valid = new Button();
    Valid.Text = "Valider";
    Valid.Click += new System.EventHandler(this.RecupQueryString);

    Label AfficheNom = new Label();
    if (Request.QueryString["name"] != null)
    {
        AfficheNom.Text = Request.QueryString["name"].ToString();
        // On récuprère notre QueryString "name" via le Request
    }

    form1.Controls.Add(EntreNom);
    form1.Controls.Add(name);
    form1.Controls.Add(Valid);
    form1.Controls.Add(AfficheNom);
}

// Cette méthode va nous permettre de créer notre QueryString
protected void RecupQueryString(object sender, EventArgs e)
{
   // On crée une variable pour plus de clarté
   string url;
   // Cette variable contiendra notre url cible avec les Query Strings que l'on
   //souhaite afficher.

   // On peut remarquer qu'à la suite de notre Url classique se trouve un "?" ce qui
   //correspond au début de notre Query String.
   // Chaque Query String est séparé par un "&" et leur syntaxe est la suivante :
   // nomDeMonIndex=MaValeurPourCetIndex
   url = "Page1.aspx?name=" +
       ((TextBox)Page.FindControl("name")).Text.ToString()+"&page=1";
   // Dans cet exemple mon QueryString "name" contiendra la valeur passé dans ma
   //Textbox "name"
   //et la QueryString "page" contiendra la valeur 1 (nous ne nous en servirons pas)

   Response.Redirect(url);
}
```

C#

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    Dim EntreNom As Label = New Label()
    EntreNom.Text = "Entrer nom"

    Dim name As TextBox = New TextBox()
    name.ID = "name"

    Dim Valid As Button = New Button()
    Valid.Text = "Valider"
    AddHandler Valid.Click, AddressOf Me.RecupQueryString

    Dim AfficheNom As Label = New Label()
    If Not Request.QueryString("name") Is Nothing Then
        AfficheNom.Text = Request.QueryString("name").ToString()
        ' On récuprère notre QueryString "name" via le Request
    End If

    form1.Controls.Add(EntreNom)
    form1.Controls.Add(name)
    form1.Controls.Add(Valid)
    form1.Controls.Add(AfficheNom)
End Sub

' Cette méthode va nous permettre de créer notre QueryString
Protected Sub RecupQueryString(ByVal sender As Object, ByVal e As EventArgs)
    ' On crée une variable pour plus de clarté
    Dim url As String
    ' Cette variable contiendra notre url cible avec les Query Strings que l'on
    ' souhaite afficher.

    ' On peut remarquer qu'à la suite de notre Url classique se trouve un "?" ce qui
    ' correspond au début de notre Query String.
    ' Chaque Query String est séparé par un "&" et leur syntaxe est la suivante :
    ' nomDeMonIndex=MaValeurPourCetIndex
    url = "Page1.aspx?name=" + CType(Page.FindControl("name"),
TextBox).Text.ToString() + "&page=1"
    ' Dans cet exemple mon QueryString "name" contiendra la valeur passé dans ma
    ' Textbox "name"
    ' et la QueryString "page" contiendra la valeur 1 (nous ne nous en servirons
pas)

    Response.Redirect(url)
End Sub
```

Aperçu :



IL faut toutefois noter que l'on ne peut pas mettre une infinité d'information dans notre URL. La majorité des navigateurs limitent la taille des URL à 2083 caractères.

Il faut donc à tout moment penser à cette limite et la gérer lors d'un passage de paramètre en *Query String*.

3 La gestion d'état côté serveur

Il existe deux moyens en ASP.NET pour pouvoir passer des informations entre les pages du côté serveur : *Application State* et *Session State*. Le premier, *Application State*, va permettre de stocker des informations qui seront accessibles dans toutes les pages Web de l'application. Il ne regarde pas quel utilisateur demande l'accès à ces informations, elles sont accessible par tout le monde. En revanche le *Session State* vérifiera l'utilisateur. La session étant spécifique à un utilisateur (avec un identifiant unique), seule la personne ayant le bon identifiant pourra accéder aux données de la session. Là encore chaque page pourra accéder aux informations de la session. Le *Session State* perdra ses informations si on l'arrête ou qu'on le vide. Par contre les deux perdront leurs données si l'application est arrêtée ou redémarrée. Pour palier à cela il va falloir définir des propriétés de profil (*profiles properties*).

3.1 Application State

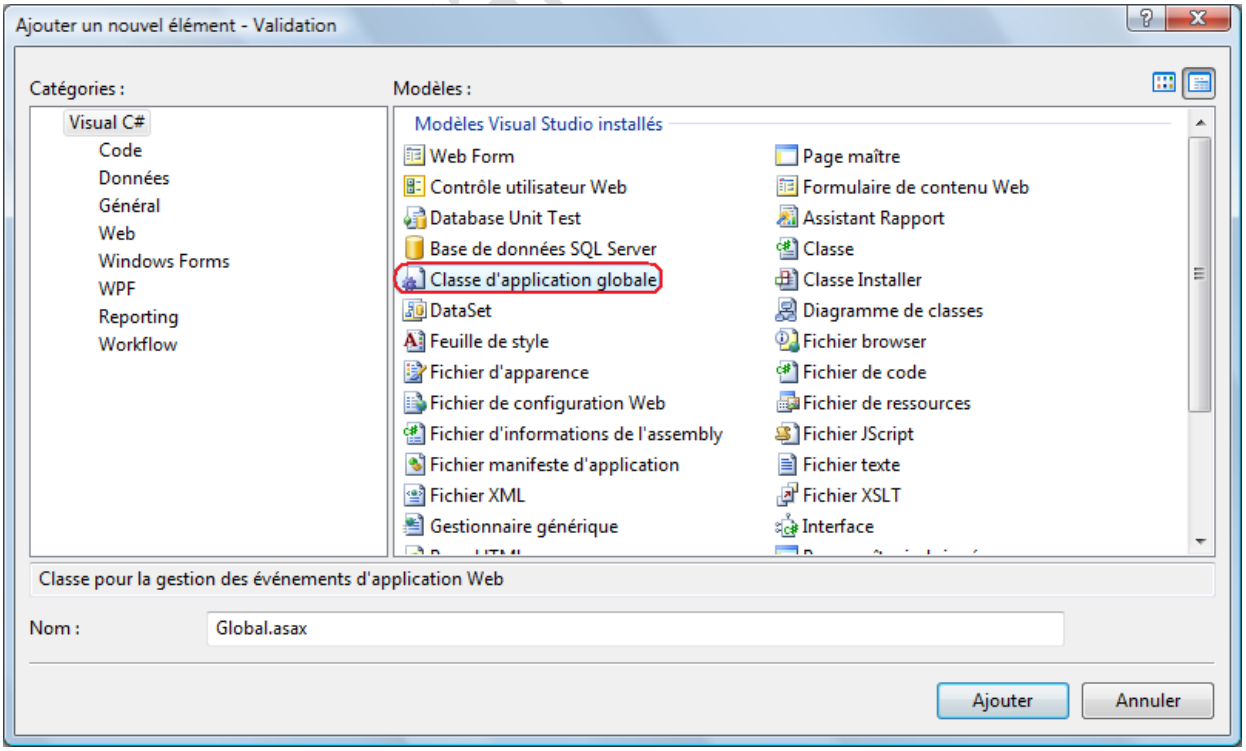
3.1.1 Présentation

C'est un moyen de stocker des données dans l'application Web, c'est-à-dire qu'elles seront accessibles par n'importe quelle page et pour n'importe quel utilisateur. Ces données seront stockées dans un dictionnaire (rappelez-vous, c'est la correspondance d'une clef avec une valeur). Le client n'aura pas accès à ces informations. Il ne pourra donc pas les changer. L'*Application State* est un bon moyen de stocker les informations qui ne sont pas spécifiques à l'utilisateur. Par exemple on peut y mettre les paramétrages par défaut qui seront utilisés si l'utilisateur n'en a pas spécifié … . Généralement on y place aussi et surtout les informations qui ne sont pas spécifiques à l'utilisateur et dont on peut avoir besoin dans plusieurs pages. En les mettant dans l'*Application State,* on n'a pas besoin de répliquer ces données.

IIS va régulièrement redémarrer l'application Web. Or comme nous l'avons dis, les données stockées dans l'*Application State* seront perdu si l'application redémarre. Une façon de procéder pour ne pas avoir de problèmes avec des données importantes et de les mettre dans les évènements *Application_Start* et *Application_End* (respectivement le démarrage de l'application et sa fermeture). Il existe un troisième évènement nommé *Application_Error* et qui correspond à une erreur dans l'application.

3.1.2 Exemple d'utilisation

Pour pouvoir utiliser les évènements de l'application il va falloir rajouter une page spéciale : une *classe d'application globale* (en anglais : *Global Application Class*) avec pour extension asax. Nous estimons que maintenant vous savez comment on rajoute une page au projet.



Vous devriez maintenant avoir une classe avec des méthodes. Parmi ces méthodes il y a celles dont nous avons parlé. Voici une image de ce que cela donne en VB.NET puisqu'il possède par défaut des commentaires sur les évènements :

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Se déclenche lorsque l'application est démarrée
End Sub

Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Se déclenche lorsque la session est démarrée
End Sub

Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
    ' Se déclenche au début de chaque demande
End Sub

Sub Application_AuthenticateRequest(ByVal sender As Object, ByVal e As EventArgs)
    ' Se déclenche lors d'une tentative d'authentification de l'utilisation
End Sub

Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    ' Se déclenche lorsqu'une erreur se produit
End Sub

Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Se déclenche lorsque la session se termine
End Sub

Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Se déclenche lorsque l'application se termine
End Sub
```

Nous allons faire un petit exemple avec un *Label* sur une page Web qui va récupérer une donnée que l'on à créé depuis l'évènement *Application_Start*. Juste après on va modifier avec l'évènement *onClick* d'un bouton cette valeur et l'afficher à nouveau dans le *Label.*

C# - Classe d'application globale

```
protected void Application_Start(object sender, EventArgs e)
{
     Application["Exemple"] = "Raté";
}
```

VB.NET – Classe d'application globale

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Se déclenche lorsque l'application est démarrée
    Application("Exemple") = "Raté"
End Sub
```

Page Web

```
<asp:Label ID="Label1" runat="server" Text="Label sans valeur" /><br />
<asp:Button runat="server" Text="Changer la valeur" onClick="Button_Click"/>
```

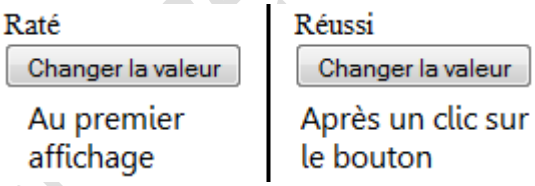
C# - Code behind de la page Web

```
protected void Page_Load()
{
    Label1.Text = Application["Exemple"].ToString();
}

protected void Button_Click(object sender, EventArgs e)
{
    Application.Lock();
    Application["Exemple"] = "Réussi";
    Application.UnLock();

    Label1.Text = Application["Exemple"].ToString();
}
```

VB.NET – Code behind de la page Web

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    Label1.Text = Application("Exemple").ToString()
End Sub

Protected Sub Button_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Application.Lock()
    Application("Exemple") = "Réussi"
    Application.UnLock()

    Label1.Text = Application("Exemple").ToString()
End Sub
```

Voici le résultat de ce code :



Dans le code behind de la page Web nous avons utilisé les méthodes *Lock* et *UnLock* de l'objet *Application*. Ces méthodes permettent de bloquer l'application. Ainsi on ne risque pas qu'une autre page en cours d'exécution vienne changer la donnée alors que l'on n'a pas fini d'écrire. Par contre il ne faut pas oublier d'utiliser *UnLock* après.

3.2 Session State

3.2.1 Présentation

Les sessions sont, elles aussi, accessibles par n'importe quelle page. Cependant chaque session possède un identifiant et pour pouvoir lire le contenu de la session il faut avoir le bon identifiant. Un utilisateur ne pourra donc accéder aux données que de sa propre session. On peut donc y stocker les préférences de chaque utilisateur puisque seuls eux peuvent y accéder. La session, par défaut, est créée automatiquement lorsqu'un utilisateur se connecte/fait une requête. Tout comme pour l'autre, les données sont stockées sous forme de dictionnaire. On y accède avec l'objet Session.

Dans l'exemple qui suit on va récupérer le pseudo rentré dans un formulaire et le mettre en session. On redirige ensuite la page vers une page de traitement qui va afficher la donnée contenu dans la session.

Page Default.aspx

```
<asp:Label runat="server" Text="Pseudo" />
<asp:TextBox ID="Pseudo" runat="server" /><br />
<asp:Button runat="server" Text="Envoyer le formulaire" OnClick="EnvoiDonnee" />
```

C# - Code behind de Default.aspx

```
protected void EnvoiDonnee(object sender, EventArgs e)
{
    Session["Pseudo"] = Pseudo.Text;
    Response.Redirect("~/Traitement.aspx");
}
```

VB.NET – Code behind de Default.aspx

```
Protected Sub EnvoiDonnee(ByVal sender As Object, ByVal e As EventArgs)
    Session("Pseudo") = Pseudo.Text
    Response.Redirect("~/Traitement.aspx")
End Sub
```

Page Traitement.aspx

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

C# - Code behind de Traitement.aspx

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["Pseudo"] != null)
        AfficherPseudo.Text = Session["Pseudo"].ToString();
}
```

VB.NET - Code behind de Traitement.aspx

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
    If Session("Pseudo") <> Nothing Then
        AfficherPseudo.Text = Session("Pseudo").ToString()
    End If
End Sub
```

Et voici ce que l'on obtient :

3.2.2 Evènements

Tout comme pour *Application State*, la session possède des évènements. Plus précisément les évènements *Session_Start* et *Session_End*. Pour les modifier il faut passer par une classe d'application globale (pour cela reportez-vous à la partie 3.1.2). Ces évènements se déclenchent respectivement, lors du démarrage d'une session, et lors de l'arrêt d'une session.

On peut, par exemple, les utiliser pour compter le nombre de visiteurs du site (en incrémentant un nombre grâce à *Session_Start*), le nombre de personne actuellement sur le site (*Session_Start* sert à augmenter le nombre et *Session_End* à le baisser) ou encore à vérifier au démarrage de la session à quel groupe l'utilisateur appartient et à charger ses préférences et droits.

3.2.3 Configuration des sessions

Il est possible de désactiver les sessions pour une page ou pour tout le site. Mais il est aussi possible de définir si on veut utiliser les sessions avec des cookies … Tout cela touche à la configuration des sessions et c'est ce que nous allons voir dans cette partie.

Commençons par la désactivation des sessions :

Comme nous l'avons dis il y a deux façons de les désactiver : sur une page ou pour le site Web entier.

Pour désactiver les sessions sur tout le site Web il faut rajouter une ligne dans le fichier de configuration *Web.config*. La ligne à ajouter est la suivante : `<sessionState mode="Off" />`

Cependant il ne doit pas être placé n'importe où. Il faut qu'il soit entre les balises system.web comme le montre le code suivant :

Fichier Web.config

```
<?xml version="1.0"?>
<configuration>
	<appSettings/>
	<connectionStrings/>
	<system.web>
		<compilation debug="true"></compilation>
		<authentication mode="Windows"/>

		<sessionState mode="Off" />
	</system.web>
</configuration>
```

Essayez maintenant de réutiliser le code que nous avons fait plus haut. Vous verrez que vous aurez une erreur lorsqu'il va essayer d'écrire le pseudo dans la session (donc lors d'un clic sur le bouton).

Pour désactiver les sessions sur une page en particulier, vous devez changer une propriété de la directive page : *EnableSessionState*.

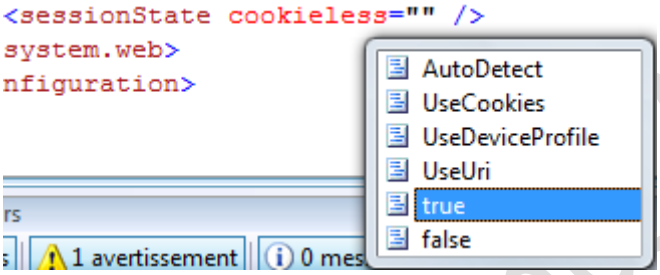
Page Default.aspx sans session

```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="Default.aspx.cs"
         Inherits="Session._Default"
        EnableSessionState="False" %>
```

Remarquez que cette propriété peut aussi être utilisée avec comme valeur ReadOnly, ce qui permettra à cette page d'accéder aux valeurs des sessions mais pas d'en créer ou d'en modifier.

Voyons maintenant comment paramétrer l'utilisation des cookies :

Par défaut, les sessions utilisent les cookies pour stocker l'identifiant de session qui va permettre à l'utilisateur d'accéder à sa session. Pour le désactiver il faut mettre dans le Web.config la propriété cookieless à true (sur le sessionState comme nous l'avions vu et comme montré sur l'image ci-dessous) :



En désactivant les cookies, ce sera l'url qui servira à passer l'identifiant de session (exemple : http://localhost:50717/(S(0hf53z3cgg1texjgou4gimit))/Traitement.aspx). On peut aussi mettre UseUri qui utilisera lui aussi l'url.

Les autres propriétés du sessionState dans le Web.config :

Mode : permet de définir le mode pour la session

InProc	Stocke les données des sessions dans la mémoire vive. Peu conseillé pour les applications Web conséquente ou sur une machine sur laquelle tournent plusieurs applications Web.
SqlServer	Stocke dans une base de données Sql. Cela permet de garder les données même si l'application Web est redémarrée. Il permet aussi l'accès à ces données pour plusieurs applications Web distante. Il faudra spécifier SqlConnectionString qui est la chaine de connexion au serveur Sql en question (voir le cours sur l'ADO.NET).
Custom	Nous permet de spécifier un provider personnalisé pour le stockage.

Off	Désactive les sessions

TimeOut : permet de spécifier la durée pendant laquelle une session est valide. Une fois ce temps dépassé, la session n'est plus considéré comme valide : on n'y a plus accès. Chaque fois que l'utilisateur fera une requête au serveur, le décompte du timeout recommencera. Donc, plus exactement, le timeout correspond à la durée d'inactivité après laquelle une session est considéré comme non valide.

3.3 Profile Properties

Il permet de stocker des données spécifiques à l'utilisateur qui navigue sur le site comme son nom, prénom, âge, le thème qu'il veut etc. Au contraire des données stockées dans l'application ou dans une session, celles stockées dans l'objet *Profile* sont persistantes. C'est-à-dire que même si l'application est redémarrée, les données existeront toujours. Pour cela, les données sont stockées dans une base de données Sql (l'objet implémente le provider SqlProfileProvider). Mais on peut lui spécifier un autre provider pour stocker par exemple dans un fichier Xml ou autres notre propre provider.

4 Conclusion

Dans ce chapitre nous avons principalement vu les différents moyens de stockage pour les données que nous utiliserons sur toutes les pages du site web ainsi que celles spécifique à l'utilisateur (nom, prénom, âge …). Pour le côté client on les utilise tous. Pour ce qui est du côté serveur on utilisera plutôt l'objet *Application* et *Session*.