Le Développement d'une Application Web

Sommaire

1 Introduction

L'un des aspects unique que nous procure ASP.NET dans le développement d'Application Web est la façon de comprendre les interactions entre le Navigateur Web et le Serveur Web. Beaucoup d'outils nous sont donnés afin de traiter les informations durant les communications client-serveur. Afin de coder l'application Web la plus optimisée qui soit, il est nécessaire de connaitre ces différents outils et de savoir les configurer.

Voici quelques exemples de compétence que vous devrez maitriser à la fin de ce chapitre :

Le développement d'une application Web :

- La détection du navigateur Web dans nos Web Forms
- Déterminer la cause d'une exception non gérée sur notre page Web
- Accéder aux entêtes d'une page Web
- Accéder aux pages encapsulées et gérer leurs contextes d'application
- Créer une page Asynchrones

Ajouter et configurer des Web Server Controls :

- Modifier le fichier de configuration du site Web via le code
- Ecrire un évènement générant des images dynamiquement affichable sur notre page Web

Afin d'appréhender ces nouvelles connaissances nous diviserons ce chapitre en deux parties :

- Le développement de Site Web
- Les requêtes d'informations

2 Le développement de site Web

Bien que nous sachions comment créer différents types de tache en ASP.NET, il est très souvent nécessaire de comprendre comment interagir directement avec les Services d'Information Internet (IIS). Ainsi, grâce à cette interaction, nous allons voir comment créer les différentes taches suivantes :

- Rattraper les exceptions non gérées
- Consulter ou modifier le fichier de configuration de notre application Web
- Lancer des taches en mode Asynchrone
- Générer dynamiquement différents types de fichier

2.1 La levée d'une exception d'une page et d'une application

La levée d'exception est le moyen le plus efficace pour rattraper des erreurs dans de petites portions de code. Par exemple, il est très courant (voire recommandé) de mettre un bloc *try/catch* entre une connexion à une base de données pour prévoir une surcharge de connections ou bien tout simplement pour palier à une impossibilité de joindre notre base de données.

Ce cours n'ayant pas pour but de vous former à l'utilisation de block try/catch nous considèrerons cette leçon comme acquise. Dans le cas contraire veuillez vous référer aux cours de C# et VB.NET.

Bien que vous prévoyiez un maximum d'erreurs dans vos blocks *try/catch* il peut arriver que cela ne suffise pas et que certaines erreurs que vous n'aviez pas prévues, se présentent. Il est possible de récupérer ces erreurs qui ne résoudront pas le problème mais pourront informer l'utilisateur du problème.

2.1.1 L'évènement *Page_Error*

Pour récupérer les erreurs sur la page, il faut créer un gestionnaire d'évènement *Page_Error* dans chaque page afin de pouvoir gérer l'évènement *Error*. Le gestionnaire d'évènement prend en paramètre un *Object* et un *EventArgs*, mais en règle générale il est inutile de se préoccuper de ces deux paramètres. Une fois à l'intérieur de notre gestionnaire d'évènement on peut utiliser les méthodes suivantes :

- *Server.GetLastError()* qui permet de retrouver la dernière erreurs en date ;
- *Server.ClearError()* qui permet d'effacer les erreurs en queue ;
- *Trace.Write()* est une fonction qui permet d'afficher le contenue de nos erreurs.

Voici un exemple type d'un gestionnaire d'évènement *Page_Error* :

C#

```
private void Page_Error(object sender, EventArgs e)
        {
            Trace.Write("ERREUR : " + Server.GetLastError().Message);
            Server.ClearError();
        }
```

VB.NET

```
Protected Sub Page_Error(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
    Trace.Write("ERREUR : " + Server.GetLastError().Message)
    Server.ClearError()
End Sub
```

Il est important de remarquer qu'il est impossible d'accéder à nos controles à partir de notre gestionnaire d'évènement.

2.1.2 L'évènement *Application_Error*

Il est également possible de lever une erreur au niveau de l'application entière. Cette autre méthode nous permet de ne plus être obligé de créer un gestionnaire d'évènement *Page_Error* sur chaque page de notre application, mais de nous renvoyer vers une même page qui traitera toutes les erreurs. Pour procéder ainsi, il faut ajouter à notre projet l'élément *Global Application Class* qui est unique pour chaque application. Tous les évènements dont nous pouvons avoir besoin sont automatiquement ajoutés par Visual Studio.

Maintenant que l'on peut accéder au fichier Global.asax il ne reste plus qu'à rediriger toutes nos erreurs vers une page de traitement.

Exemple :

C# de Global.asax

```
protected void Application_Error(object sender, EventArgs e)
        {
            Server.Transfer("ErrorPage.aspx");
           // ErrorPage.aspx est bien entendu une page que vous
            // aurez créée dans laquel nous récupèrerons
            // les messages d'erreurs pour les afficher
           // grace aux méthode vu précédement
        }
```

C# de Global.asax

```
Protected Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    Server.Transfer("ErrorPage.aspx")
    ' ErrorPage.aspx est bien entendu une page que vous
    ' aurez créée dans laquel nous récupèrerons
    ' les messages d'erreurs pour les afficher
    ' grace aux méthode vu précédement
End Sub
```

2.2 Intervenir dynamiquement sur le fichier de configuration Web.config

Vous avez surement du fréquemment toucher au fichier de configuration Web.config de façon manuelle pour activer ou désactiver les *ViewStates*, ou bien activer le mode Debug dans Visual Studio (demandé automatiquement à la première compilation de chaque projet). Il existe cela dit une manière dynamique (en passant par le *CodeBehind*) pour modifier ce fichier de configuration. Ce genre de manipulation peut s'avérer très utile dans le cas de déploiement assisté de votre application Web, ou bien dans une interface d'administrateur etc….

Au cours de cette partie nous allons voir comment effectuer de telles modifications. En premier lieu, il nous faut utiliser l'objet *System.Configuration.Configuration* afin de lire et écrire dans le *Web.config*. Dans le but de créer un objet *Configuration* de notre application courante, il vous faudra faire appel à *WebConfigurationManager*.

A partir de cet objet *Configuration*, il nous est possible d'accéder aux méthodes *GetSection* et *GetSectionGroup* qui vont nous permettre de lire dans nos sections de notre fichier de configuration. Il est nécessaire que l'utilisateur en cours ait les autorisations de lecture sur tous les fichiers de configuration. Une fois vos actions finies, deux choix s'offrent à vous :

- Grace à la méthode *Save* vous pourrez sauvegarder vos changements effectués, il faut toutefois posséder les droits de modifications pour que cette action soit valide.
- Grace à la méthode *SaveAs* vous pourrez sauvegarder vos changements dans un autre fichier de configuration, dans ce cas là, il faut en plus les droits de création de fichier.

Il est préférable d'utiliser la méthode *SaveAs* lorsque l'on veut spécifier une configuration bien précise à un sous-dossier par exemple.

Voici une portion de code montrant comment modifier un fichier de configuration. Pour que ce code fonctionne, il ne faut pas oublier d'ajouter le namespace *System.Web.Configuration*.

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    //Tout d'abord il nous faut récupérer la configuration Web de notre application
    System.Configuration.Configuration config =
System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~/Test");

    //Ensuite récupérons la section désirée
    AuthenticationSection section =
       (AuthenticationSection)config.GetSection("system.web/authentication");

    //On cré un Label pour stocker les informations contenue dans notre section
    // et que l'on souhaitera afficher.
    Label Label1 = new Label();
    Label1.Text = section.Mode.ToString();

    form1.Controls.Add(Label1);
}
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
    ' Tout d'abord il nous faut récupérer la configuration Web de notre application
    Dim config As System.Configuration.Configuration =
System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~/Test")

    ' Ensuite récupérons la section désirée
    Dim section As Web.Configuration.AuthenticationSection =
        config.GetSection("system.web/authentication")

    ' On cré un Label pour stocker les informations contenue dans notre section
    ' et que l'on souhaitera afficher.
    Dim Label1 As Label = New Label()
    Label1.Text = section.Mode.ToString()

    form1.Controls.Add(Label1)
End Sub
```

Chaque élément du Web.config dispose de sa propre classe. Voici un tableau détaillant les plus utilisés.

Class	Section de Configuration dans <system.web>	Description
AuthenticationSection	<authentication>	Configure l'authentification pour une page Web
AnonymousIdentificationSection	<anonymousIdentification>	Configure l'identification anonyme pour les utilisateurs non authentifiés
AuthorizationSection	<authorization>	Configure une autorisation d'application Web
CacheSection	<cache>	Configure les paramètres de cache globaux pour une application ASP.NET
CompilationSection	<compilation>	Définit des paramètres de configuration qui sont utilisés pour prendre en charge l'infrastructure de compilation d'applications Web
CustomErrorsSection	<customErrors>	Configure les erreurs personnalisées ASP.NET
DeploymentSection	<deployement>	Définit des paramètres de configuration qui

		sont utilisés pour prendre en charge le déploiement d'une application Web
GlobalizationSection	<globalization>	Définit des paramètres de configuration qui sont utilisés pour prendre en charge l'infrastructure de globalisation d'applications Web
HealthMonitoringSection	<heathMonitoring>	Configure les profiles ASP.NET afin de déterminer comment les évènements du health-monitoring sont envoyé vers le gestionnaire d'évènement
HostingEnvironmentSection	<hostingEnvironment>	Définit des paramètres de configuration qui contrôlent le comportement de l'environnement d'hébergement de l'application
HttpCookiesSection	<httpCookies>	Configure des propriétés pour les cookies utilisés par une application Web
HttpHandlersSection	<handlersSection>	Configure un gestionnaire HTTP pour une application Web
HttpRuntimeSection	<httpRuntime>	Configure le runtime HTTP ASP.NET
IdentitySection	<identity>	Configure l'identité d'une application Web
MachineKeySection	<manchineKey>	Définit les paramètres de configuration qui contrôlent la génération de clés et les algorithmes utilisés dans le chiffrement, le déchiffrement et les opérations MAC (Media Access Control) dans l'authentification Windows Forms, la validation d'état d'affichage et l'isolation d'application des états de session
MembershipSection	<membership>	Définit les paramètres de configuration pour la prise en charge de l'infrastructure permettant de configurer et de gérer les détails d'appartenance (membership)
OutputCacheSection	<outputCache>	Configure le cache de sortie pour une application Web
PagesSection	<pages>	Fournit l'accès par programme à la section de l'élément pages (Schéma des paramètres ASP.NET) du fichier de configuration
ProfileSection	<profile>	La classe *ProfileSection* permet d'accéder au contenu de la section profile du fichier de configuration et de le modifier par programme.
SecurityPolicySection	<securityPolicy>	Définit des paramètres de configuration qui sont utilisés pour prendre en charge l'infrastructure de sécurité d'une application Web
SessionPageStateSection	<sessionPageState>	Configure la section sessionPageState
SessionStateSection	<sessionState>	Configure l'état de session pour une application Web

SiteMapSection	<siteMap>	Définit des paramètres de configuration utilisés pour prendre en charge l'infrastructure pour la configuration, le stockage et le rendu de la navigation de site
SqlCacheDependencySection	<sqlCacheDependency>	Configure les dépendances de cache SQL pour une application ASP.NET
TraceSection	<trace>	Configuration du service de trace ASP.NET
TrustSection	<trust>	Configure le niveau de sécurité d'accès du code appliqué à une application
WebControlsSection	<webControls>	Configure la section webControls
WebPartsSection	<webParts>	Fournit l'accès par programme à la section du fichier de configuration webParts
XhtmlConformanceSection	<xhtmlConformance>	Configure la section xhtmlConformance

Une fois que vous aurez créez une instance d'une de ces classes, vous pouvez utiliser leurs méthodes ainsi que leurs propriétés pour lire et écrire des informations dans le fichier de configuration.

Il est possible d'afficher nos modifications personnalisées dans un *Label* comme peut le montrer la ligne de code suivante :

C#

```
Label1.Text = WebConfigurationManager.AppSettings["Test"];
```

VB.NET

```
Label1.Text = Web.Configuration.WebConfigurationManager.AppSettings("Test")
```

Ici *Test* représente le nom de notre application/solution.

Il est aussi possible d'accéder dynamiquement au *Connection Strings* en utilisant la collection *WebConfigurationManager.ConnectionStrings.*

Exemple :

C#

```
Label1.Text = WebConfigurationManager.ConnectionStrings["MaBase"].ConnectionString;
```

VB.NET

```
Label1.Text =
Web.Configuration.WebConfigurationManager.ConnectionStrings("MaBase").ConnectionString
```

La méthode statique *WebConfigurationManager* est le moyen le plus efficace pour consulter la configuration du site car elle prend en compte la hierarchie entière.

En revanche si vous souhaitez effectuer des changements dans cette configuration il vous faudra choisir un emplacement spécifique pour votre configuration. Dans ce but, vous allez devoir créer une instance de l'objet *Configuration* dont la syntaxe est un peu particulière.

Pour créer une instance qui touchera la racine de votre Web.Config (qui s'appliquera à toute votre application Web) il faut appeler la méthode statique *WebConfigurationManager.OpenWebConfiguration* et lui passer « ~ » en paramètre. Ensuite, il faut utiliser l'objet *Configuration* pour créer des objets destinés à des sections individuelles.

Une fois vos opérations terminées, il suffit d'appeler la méthode *Configuration.Save()* pour sauvegarder vos modifications.

Exemple :

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    //Ce code rajoute <trace enabled="true" /> à notre fichier de Web.config

    Configuration rootConfig =
        System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~");
    //Il est possible de passer en paramètre le chemin de notre application
    //Exemple : WebConfiguration.OpenWebConfiguration("/Test")

    System.Web.Configuration.TraceSection trace =
        (System.Web.Configuration.TraceSection)rootConfig.GetSection("system.web/tra
        ce");
    trace.Enabled = true;
    rootConfig.Save();
}
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
    ' Ce code rajoute <trace enabled="true" /> à notre fichier de Web.

    Dim rootConfig As System.Configuration.Configuration =
        System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~")
    ' Il est possible de passer en paramètre le chemin de notre application
    ' Exemple : WebConfiguration.OpenWebConfiguration("/Test")

    Dim trace As System.Web.Configuration.TraceSection =
        CType(rootConfig.GetSection("system.web/trace"),
        System.Web.Configuration.TraceSection)
    Trace.Enabled = True
    rootConfig.Save()
End Sub
```

Il existe trois façons différentes de sauvegarder notre configuration que l'on utilise grace à l'énumération *ConfigurationSaveMode* :

Full : Toutes les propriétés sont écrites dans le fichier de configuration. Très utilisé pour le deplacement de l'application d'une machine à une autre.

Minimal : Seules les propriétés qui diffèrent des valeurs héritées sont écrites dans le fichier de configuration.

Modified : Seules les propriétés modifiées seront écrites dans le fichier de configuration, et ce, même lorsque la valeur est la même que celle dont elle hérite.

Pour créer un nouveau fichier de configuration il faut appeler la méthode *Configuration.SaveAs()* et définir son emplacement.

Exemple :

C#

```
Configuration config = WebConfigurationManager.OpenWebConfiguration("~");
config.SaveAs(Server.MapPath("Test.Web.config"), ConfigurationSaveMode.Full, true);
//Rappelons que Server.MapPath() nous permet de donner un chemin relatif.
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
    Dim rootConfig As System.Configuration.Configuration =
        System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~")
    rootConfig.SaveAs(Server.MapPath("Test.Web.config"), ConfigurationSaveMode.Full,
        True)
    ' Rappelons que Server.MapPath() nous permet de donner un chemin relatif.
End Sub
```

2.3 Créer des pages Web Asynchrone

Jusqu'à présent nous ne nous sommes attachés qu'à la conception de notre application Web, désormais nous allons aborder la notion de performance (rapidité) de notre application. L'introduction à ce concept est très particulière car il sera difficile de constater cette amélioration lors du développement. Rappelons que nous sommes les seuls à effectuer des requêtes sur nos pages Web durant cette étape. La différence ne sera notable que lorsque l'application sera en ligne et qu'une multitude d'utilisateurs effectuerons des requêtes en simultané.

Faire de la programmation en mode asynchrone permet entre autre de créer une application à plusieurs threads, c'est-à-dire une application disposant de plusieurs processus légers s'exécutant en même temps (en parallèle). L'avantage de ce type de programmation est que faire tourner plusieurs petits processus augmente de façon considérable la performance de notre application.

La programmation asynchrone peut s'avérer relativement compliquée au premier abord, mais elle est parfois indispensable dans le cas d'accès à une base de données par exemple, afin de diminuer le temps d'attente lors de son accès.

2.3.1 Pourquoi utiliser le mode Asynchrone

Si vous avez déjà fait de la programmation en WinForm Application et que vous avez déjà abordé les threads, il faudra bien comprendre que les objectifs ne sont absolument pas les mêmes. En effet, tandis que l'un des objectifs du thread est de lancer un petit processus en parallèle en attendant une réponse de l'utilisateur, par exemple dans le cas d'un script de Tchat, l'objectif principal de l'asynchrone en Web est d'alléger au maximum la charge serveur afin d'avoir un accès aux données et par conséquent un affichage de notre page Web le plus rapide possible. Car, contrairement au WinForm (dans le cas d'une application avec dialogue réseau) où chacun dispose de sa propre partie cliente déjà installé sur sa propre machine, plusieurs requêtes sont fréquemment envoyées par plusieurs utilisateurs en même temps.

De base, ASP.NET et IIS ne peuvent effectuer le rendu que d'un nombre limité de page simultanément. Par conséquent, plus le pool de thread est consommé plus les performances diminuent tel un entonnoir. Une fois ce pool complètement consommé, les autres pages sont mises en attente tant que le rendu des pages en cours n'est pas terminé.

L'activation et la bonne gestion du mode asynchrone permet d'augmenter le nombre de rendu simultané et donc d'améliorer les performances de notre application Web.

2.3.2 Activation et utilisation du mode Asynchrone

Nous allons expliquer comment utiliser le mode Asynchrone en trois étapes. On estimera que vous saurez implémenter les espaces de nommage adéquat.

Etape 1 :

Pour commencer, il faut ajouter `Async="true"` à la directive page dans laquelle on inclura des évènements en asynchrone.

ASP.NET

```
<%@ Page Language="c#" Async="true" CodeBehind="Default.aspx.cs"
Inherits="Test._Default" %>
```

Cette directive indique au compilateur qu'il faut que la page implémente *IHttpAsyncHandler* au lieu de *IHttpHandler*. Cette interface permet d'implémenter des fonctionnalités de traitements asynchrones à la page.

Etape 2 :

Dans le CodeBehind de notre page, nous allons créer nos taches qui vont se lancer en mode asynchrone dans notre *Page_Load* par exemple.

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    //Nous allons commencer par créer nos taches
    PageAsyncTask myTask = new PageAsyncTask(new BeginEventHandler(BeginFirstTask),
                                             new EndEventHandler(EndFirstTask),
                                             new EndEventHandler(TimeoutFirstTask),
                                             null);
    //On peut remarquer que le constructeur PageAsyncTask prend en paramètre :
    // - un BeginEventHandler : gère l'évènement de début de notre tache ;
    // - un EndEventHandler : gère l'évènement de fin de notre tache ;
    // - un TimoutException : gère l'évènement en cas de requète trop longue ;
    // - un Objet state : mis à null par défaut

    //Enregistrons notre tache maintenant :
    Page.RegisterAsyncTask(myTask) ;
}
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
    ' Nous allons commencer par créer nos taches
    Dim myTask As PageAsyncTask = New PageAsyncTask(New BeginEventHandler(AddressOf
        BeginFirstTask), New EndEventHandler(AddressOf EndFirstTask), New
        EndEventHandler(AddressOf TimeoutFirstTask), DBNull.Value)
    ' On peut remarquer que le constructeur PageAsyncTask prend en paramètre :
    ' - un BeginEventHandler : gère l'évènement de début de notre tache ;
    ' - un EndEventHandler : gère l'évènement de fin de notre tache ;
    ' - un TimoutException : gère l'évènement en cas de requête trop longue ;
    ' - un Objet state : mis à null par défaut

    ' Enregistrons notre tache maintenant :
    Page.RegisterAsyncTask(myTask)
End Sub
```

Notons qu'aucune action n'a été définie pour nos taches asynchrones pour le moment. Nous allons cependant expliquer comment va être traité ce bout de code.

1. Un premier thread est donné à la requête. La page est traitée en entier jusqu'au *Page_PreRenderComplete* ;
2. La page liste toutes les tâches et les laisse, les une après les autres, envoyer leur requête asynchrone, puis elle rend le thread au pool ASP.NET ;
3. Lorsque les requêtes sont en train d'être traités aucun thread du pool ASP.NET n'est consommé ;
4. Quand ces requêtes « reviennent », ASP.NET redonne un thread à la page, qui exécute les fonctions de retour ou de timeout, puis fini l'exécution normale de la page (le *Page_PreRenderComplete* et la suite).

On peut forcer ASP.NET à effectuer toute cette mécanique avant le *Page_PreRenderComplete* en appelant explicitement la méthode *Page.ExecuteRegisteredAsyncTasks*.

Etape 3 :

Nous allons maintenant gérer nos évènements correspondant à chacune de nos taches.

BeginFirstTask :

C#

```
public partial class _Default : System.Web.UI.Page
{
    //Il nous faut créer des attirbuts par lesquels on va passer pour
    // communiquer entre nos évènements
    private string dataBase = @"Data Source=\SQLEXPRESS;
                                Initial Catalog=dbTest;
                                Integrated Security=True;
                                Pooling=False;
                                Asynchronous Processing=true";
    //Changer la valeur de votre attribut en fonction de votre base de données
    //Il ne faut surtout pas oublier d'ajouter "Asynchronous Processing=true"
    // pour accéder à notre base de données en mode asynchrone

    private SqlCommand Commande;

    protected void Page_Load(object sender, EventArgs e)
    {
       ➔ Voir plus haut
    }

    //Evènement du début de notre tache
    IAsyncResult BeginFirstTask(object sender, EventArgs e, AsyncCallback cb, object
state)
    {
       //Nous allons nous connecter à une base de données
       //On peut admettre que cette opération est longue si une grand nombre
       // d'utilisateur est connecté en même temps
       SqlConnection MaConnection = new SqlConnection(dataBase);
       //Il faut bien sur avoir crée une base de données contenant la table sur
      // laquelle nous effectuons notre requête plus bas.

       MaConnection.Open();

       SqlCommand MaCommande = new SqlCommand(
       "SELECT * FROM Users", MaConnection);
       //Nous allons stocker notre sql Commande dans nos
       // attributs pour les récupérer dans nos autres taches
       Commande = MaCommande;

       //On termine par renvoyé l'execetion de notre évènement
       // qui va être exécuter en mode asynchrone.
       return Commande.BeginExecuteReader(cb, state);
    }
}
```

VB.NET

```
Partial Public Class _Default
    Inherits System.Web.UI.Page

    ' Il nous faut créer des attirbuts par lesquels on va passer pour
    ' communiquer entre nos évènements
    Private dataBase As String = "Data Source=\SQLEXPRESS;Initial
Catalog=dbTest;Integrated Security=True;Asynchronous Processing=true"
    ' Changer la valeur de votre attribut en fonction de votre base de données
    ' Il ne faut surtout pas oublier d'ajouter "Asynchronous Processing=true"
    ' pour accéder à notre base de données en mode asynchrone

    Private Commande As SqlCommand


    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
        ➔ Voir plus haut
    End Sub


    ' Evènement du début de notre tache
    Function BeginFirstTask(ByVal sender As Object, ByVal e As EventArgs, ByVal cb
As AsyncCallback, ByVal state As Object) As IAsyncResult
        ' Nous allons nous connecter à une base de données
        ' On peut admettre que cette opération est longue si une grand nombre
        ' d'utilisateur est connecté en même temps
        Dim MaConnection As SqlConnection = New SqlConnection(dataBase)
        ' Il faut bien sur avoir crée une base de données contenant la table sur

        ' laquelle nous effectuons notre requête plus bas.

        MaConnection.Open()

        Dim MaCommande As SqlCommand = New SqlCommand("SELECT * FROM Users",
             MaConnection)
        ' Nous allons stocker notre sql Commande dans nos
        ' attributs pour les récupérer dans nos autres taches
        Commande = MaCommande

        ' On termine par renvoyé l'execetion de notre évènement
        ' qui va être exécuter en mode asynchrone.
        Return Commande.BeginExecuteReader(cb, state)
    End Function
End Class
```

EndFirstTask :

Afin de lier nos évènements à notre base de données et les afficher nous allons créer une *GridView* sur notre page ASPX.

ASPX

```
<form id="form1" runat="server">
    <asp:GridView ID="MonRepeateur" runat="server">
    </asp:GridView>
</form>
```

Gérons maintenant l'évènement après **BeginFirstTask**.

C#

```
void EndFirstTask(IAsyncResult ar)
{
   //Lions notre base de données à notre GridView
   MonRepeateur.DataSource = Commande.EndExecuteReader(ar);
   MonRepeateur.DataBind();
}
```

VB.NET

```
Protected Sub EndFirstTask(ByVal ar As IAsyncResult)
    'Lions notre base de données à notre GridView
    MonRepeateur.DataSource = Commande.EndExecuteReader(ar)
    MonRepeateur.DataBind()
End Sub
```

TimeoutFirstTask :

Pour finir, nous allons gérer l'évènement timeout. Par exemple on peut créer un *Label* sur notre page ASPX dont la propriété texte sera gérée dans le CodeBehind.

ASPX

```
<form id="form1" runat="server">
    <asp:GridView ID="MonRepeateur" runat="server">
    </asp:GridView>
    <asp:Label ID="MonLabel" runat="server"></asp:Label>
</form>
```

C#

```
//Evènement de Timeout
void TimeoutFirstTask(IAsyncResult ar)
{
   MonLabel.Text = "Erreur de connection";
}
```

VB.NET

```
' Evènement de Timeout
Protected Sub TimeoutFirstTask(ByVal ar As IAsyncResult)
    MonLabel.Text = "Erreur de connection"
End Sub
```

Code Source final :

ASPX

```
<body>
    <form id="form1" runat="server">
    <asp:GridView ID="MonRepeateur" runat="server">
    </asp:GridView>
    <asp:Label ID="MonLabel" runat="server"></asp:Label>
    </form>
</body>
```

C#

```
public partial class _Default : System.Web.UI.Page
{
    private string dataBase = @"Data Source=\SQLEXPRESS;
                                Initial Catalog=dbTest;
                                Integrated Security=True;
                                Pooling=False;
                                Asynchronous Processing=true";
    private SqlCommand Commande;

    protected void Page_Load(object sender, EventArgs e)
    {
        PageAsyncTask myTask = new PageAsyncTask(
                                new BeginEventHandler(this.BeginFirstTask),
                                new EndEventHandler(this.EndFirstTask),
                                new EndEventHandler(this.TimeoutFirstTask),
                                null);
        Page.RegisterAsyncTask(myTask);
    }

    IAsyncResult BeginFirstTask(object sender, EventArgs e,
                                AsyncCallback cb, object state)
    {
        SqlConnection MaConnection = new SqlConnection(dataBase);
        MaConnection.Open();
        SqlCommand MaCommande = new SqlCommand("SELECT * FROM Users", MaConnection);
        Commande = MaCommande;
        return Commande.BeginExecuteReader(cb, state);
    }

    void EndFirstTask(IAsyncResult ar)
    {
        MonRepeateur.DataSource = Commande.EndExecuteReader(ar);
        MonRepeateur.DataBind();
    }

    void TimeoutFirstTask(IAsyncResult ar)
    {
        MonLabel.Text = "Erreur de connection";
    }
}
```

VB.NET

```
Partial Public Class _Default
    Inherits System.Web.UI.Page

    Private dataBase As String = "Data Source=\SQLEXPRESS;Initial
Catalog=dbTest;Integrated Security=True;Pooling=False;Asynchronous Processing=true"
    Private Commande As SqlCommand

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load

        Dim myTask As PageAsyncTask = New PageAsyncTask(New
BeginEventHandler(AddressOf BeginFirstTask), New EndEventHandler(AddressOf
EndFirstTask), New EndEventHandler(AddressOf TimeoutFirstTask), DBNull.Value)

        Page.RegisterAsyncTask(myTask)
    End Sub

    Function BeginFirstTask(ByVal sender As Object, ByVal e As EventArgs, ByVal cb
As AsyncCallback, ByVal state As Object) As IAsyncResult
        Dim MaConnection As SqlConnection = New SqlConnection(dataBase)

        MaConnection.Open()

        Dim MaCommande As SqlCommand = New SqlCommand("SELECT * FROM Users",
MaConnection)

        Commande = MaCommande

        Return Commande.BeginExecuteReader(cb, state)
    End Function

    Protected Sub EndFirstTask(ByVal ar As IAsyncResult)
        MonRepeateur.DataSource = Commande.EndExecuteReader(ar)
        MonRepeateur.DataBind()
    End Sub

    Protected Sub TimeoutFirstTask(ByVal ar As IAsyncResult)
        MonLabel.Text = "Erreur de connection"
    End Sub

End Class
```

2.4 Générer dynamiquement des images

Lors de l'affichage d'une image dans notre application Web, IIS lit simplement le fichier image du système de fichier, et les renvoie directement au navigateur Web. Cependant il peut arriver que dans certains cas on ait besoin qu'une image soit générée automatiquement, par exemple pour des miniatures de photos ou pour afficher un graphique résultant de statistique ou sondage.

Il est bien entendu possible de générer des miniatures ou des graphiques à l'avance et de les afficher, et bien que cette solution soit la plus légère pour le serveur, nous sommes fréquemment obligés de générer nous même ces images. ASP.NET nous donne justement des outils qui vont nous permettre de créer ces miniatures et ces graphiques.

Dans cette partie nous allons voir comment utiliser quelques méthodes comprises dans les espaces de nommage *System.Drawing* et *System.Drawing.Imaging.* Afin de les illustrer nous allons vous montrer comment créer la miniature d'une image.

Il existe une méthode appartenant aux objets *Image* qui va nous permettre, de façon très simple, de générer des miniatures dynamiquement : la méthode *GetThumbnailImage()* prenant en paramètre une largeur, une hauteur, un callback, et un *IntPtr* qu'il faut obligatoirement mettre à zéro.

Exemple :

C#

```
public bool ThumbnailCallback()
        // Necessaire pour la génération de miniatures
        {
            return false;
        }

public void CreerMiniature()
        {
            Image.GetThumbnailImageAbort myCallback = new
                                Image.GetThumbnailImageAbort(ThumbnailCallback);
            // -> Délégué Image.GetThumbnailImageAbort. Dans GDI+ version 1.0, le
                  délégué n'est pas utilisé.
            //    Vous devez toutefois créer un délégué et passer une référence à ce
                  délégué dans ce paramètre.
            Bitmap bmp = new Bitmap(@"C:\monImage.jpg");
            // -> On crée un nouvel objet Bitmap pointant sur notre fichier chargé
            Image maMiniature =
                        bmp.GetThumbnailImage(100,
                                              100,
                                              myCallback,
                                              IntPtr.Zero);
            // -> On crée notre miniature
         }
```

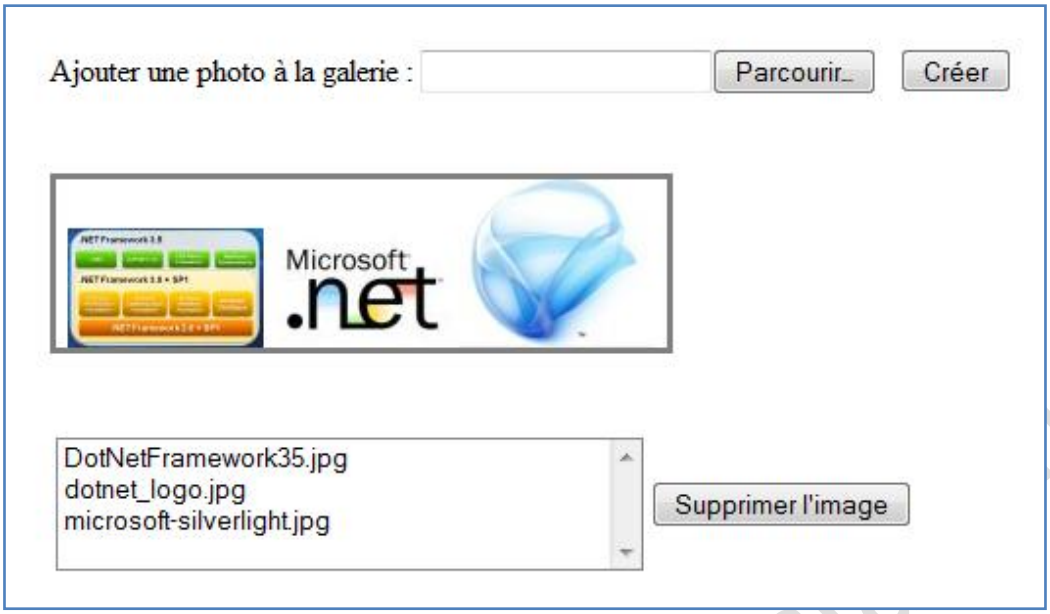
VB.NET

```
Public Function ThumbnailCallback() As Boolean
    ' Necessaire pour la génération de miniatures
    Return False
End Function

Public Sub CreerMiniature()
    Dim myCallback As Drawing.Image.GetThumbnailImageAbort = New
Drawing.Image.GetThumbnailImageAbort(AddressOf ThumbnailCallback)
    ' -> Délégué Image.GetThumbnailImageAbort. Dans GDI+ version 1.0, le
délégué n'est pas utilisé.
    '    Vous devez toutefois créer un délégué et passer une référence à ce
         délégué dans ce paramètre.
    Dim bmp As Drawing.Bitmap = New Drawing.Bitmap("C:\monImage.jpg")
    ' -> On crée un nouvel objet Bitmap pointant sur notre fichier chargé
    Dim maMiniature As Drawing.Image = bmp.GetThumbnailImage(100, 100, myCallback,
IntPtr.Zero)
    ' -> On crée notre miniature
End Sub
```

Afin de bien observer l'utilité d'une telle fonction nous vous proposons un TP à réaliser sur la création d'une galerie d'image. Ce TP se trouve en annexe de cette leçon dans le dossier « Chapitre 8 TP ».

Voici un apperçu de la page que vous serez amené à créer au cours de ce TP :



3 Récupération d'information

Nous allons voir comment il est possible dans une application Web ASP.NET de récupérer et modifier les informations relatives au serveur, au client et autres.

3.1 En général

En ASP.NET nous avons plusieurs objet qui nous permettent de récupérer des informations et pour certaine de les définir et/ou modifier. Ces objets contiennent des informations sur tout ce qui touche la communication entre le client (navigateur) et le serveur. Voyons ces objets un peu plus en détails :

Objets	Définition
Application	Cet objet permet l'accès à un cache de l'application qui peut être lu et écrit par n'importe quel utilisateur (indifféremment des sessions qu'ils possèdent). On s'en sert pour stocker des informations qui ne sont pas spécifiques à l'utilisateur mais qui serviront pour tous. Ce qui est stocké dans ce dictionnaire est accessible de n'importe quel endroit de l'application.
Context	Cet objet permet d'accéder à tout ce qui correspond au contexte actuel de l'application. Pour faire simple c'est un autre moyen d'accéder aux objets Session, Request, Response, Server, etc. …. Ainsi qu'aux erreurs qui ont pu se produire dans l'application.
Request	Il nous permet d'accéder à tout ce qui touche la requête de l'utilisateur. Il nous donne donc l'accès à tout ce qui a été envoyé par le navigateur : header, cookies, query string, etc.
Response	Celui-ci vient à la suite du Request puisqu'il nous permet d'accéder à tout ce qui va être envoyé au navigateur du client à la suite de sa requête. On pourra donc

	modifier les headers, ajouter du texte à la page, insérer des cookies, etc.
Server	Fourni l'accès à la dernière erreur du serveur ainsi qua des méthodes du serveur. Entre autre il permet d'effectuer une redirection sur le serveur (c'est-à-dire que le client aura le nom de la page qu'il a demandé qu'il en verra une autre : le serveur lui a envoyé la page sur laquelle on aura redirigé). Il permet aussi de récupérer la racine du serveur avec sa méthode MapPath.
Session	Il nous permet de travailler sur la session de l'utilisateur. Un utilisateur possédant une session possède aussi un identifiant de session. Celui-ci est stocké sur le client. Cet objet va nous permettre de définir, entre autres, de quel manière il sera stocké (cookies, url, etc.). Pour exemple on peut ajouter, éditer ou supprimer des cookies dans la session d'un utilisateur.
Trace	Cet objet permet de récupérer toutes les données disponibles et de les afficher. Par exemple on peut afficher toutes les variables session ou application avec leurs valeurs. Ou encore toutes les variables du serveur et autres.

Nous avons déjà vu dans le chapitre précédent (Gestion d'état) les objets *Session* et *Application*. Nous allons donc voir plus en détails les objets *Response*, *Request*, Server et *Context*.

3.1.1 Response

Méthodes	Définition
BinaryWrite	Ecrit des caractères binaire dans le flux de réponse HTTP.
AppendHeader	Ajoute des headers dans le flux de réponse HTTP.
Clear	Vider le flux de réponse HTTP.
ClearContent	Supprimer le contenu du flux HTTP. Cela n'inclu pas les headers.
End	Ferme le flux et envoie la page à l'utilisateur.
Flush	Envoie directement la page à l'utilisateur sans fermer le flux. Ce système est utilisé par exemple pour envoyer une page en partie. par exemple pour le traitement d'un formulaire on peut écrire dans le flux avec la méthode Write pour écrire "En cours de traitement". Ensuite on appel cette méthode pour envoyer le flux alors que le traitement n'est pas fini.
Redirect	Permet de rediriger l'utilisateur vers une autre page. Il existe aussi la méthode Server.Transfer.
TransmitFile	Ecrit un fichier dans le flux HTTP sans le mettre en buffer.
Write	Permet d'écrire dans le flux HTTP avec buffer.
WriteFile	Permet de rediriger l'utilisateur.
WriteSubstitution	Remplace un string dans le flux de réponse.

Propriétés	Définition
Cookies	Permet de lire / écrire un cookie.
Buffer	Si cette propriété est à True, la réponse sera mise en buffer avant d'être envoyé à l'utilisateur. Si elle est à false, elle ne sera pas bufférisé et sera envoyé en morceau à l'utilisateur. Utile si vous devez envoyer une grande quantité de donnée (il vaut mieux l'envoyer en morceau).
Cache	Permet d'obtenir les options de mise en page (stratégie) de la page Web. Entre autre il y a le délai d'expiration.
Expires	Délai en minutes après lequel le navigateur doit arrêter la mise en cache de la page.
ExpiresAbsolute	Il correspond au délai durant lequel, si l'utilisateur accède à nouveau à cette page, la page mise en cache sera réaffiché. Ce délai dépassé, la page n'est plus valide.
Status / StatusCode	Permet de lire ou écrire le code HTTP du statut de la requête. Pour cela ce reporter au chapitre 1.

3.1.2 Request

Méthodes	Définition
MapPath	Détermine un chemin, une adresse, vers un fichier en absolu (adresse complète) à partir de la racine du serveur. Par exemple avec Default.aspx, il va retourner le chemin absolu sur le système de ce fichier.
SaveAs	Permet de sauvegarder la requête dans le fichier spécifié.
ValidateInput	Génère une exception si les données envoyées par l'utilisateur présentent un risque de danger. Par défaut elle est activée.

Propriétés	Définition
ApplicationPath	Récupère le chemin sur le système de la racine de l'application.
Browser	Récupère les informations relative au navigateur de l'utilisateur comme son nom, sa version, ses fonctionnalités etc.
ClientCertificate	Si le client fourni un certificat de sécurité, il le

	récupère.
Cookies	Permet de lire / écrire un cookie.
FilePath	Récupère le chemin virtuel de la requête courante.
Files	Récupère une collection de fichier envoyé par l'utilisateur.
Headers	Récupère une collection contenant les headers de la requête.
HttpMethod	Récupère la méthode de transfert de données utilisé par le client comme Get, Post ou Head.
IsAuthenticated	Récupère un booléen qui défini si l'utilisateur est authentifié ou non.
isLocal	Récupère un booléen qui définit si l'utilisateur utilise un ordinateur sur le réseau local ou non.
IsSecureConnection	Récupère un booléen qui permet de dire si la connexion utilisé est sécurisé ou non (elle est sécurisé quand on passe par HTTPS).
LogonUserIdentity	Récupère l'objet Windowsidentity qui représente l'utilisateur courant.
Params	Récupère une collection qui inclut les items QueryString, Form, ServerValidation et Cookies.
Path	Donne le chemin virtuel de la requête courante.
PhysicalApplicationPath	Fourni le chemin physique vers le dossier racine de l'application.
PhysicalPath	Récupère le chemin physique de la requête courante.
QueryString	Récupère une collection contenant les variables se trouvant dans l'url.
RawUrl / Url	Donne l'url de la requête courante.
TotalBytes	Récupère la taille de la requête.
UrlReferrer	Récupère l'url de la page précédente.
UserAgent	Récupère une chaine de caractères qui caractérise le navigateur (comprend aussi le nom du navigateur).
UserHostAddress c	Récupère l'adresse IP du lient.
UserHostName	Récupère le nom du DNS du client.
UserLanguages	Récupère la langue de l'utilisateur. Correspond à un string stocké dans le navigateur du client.

3.1.3 Server

Méthodes	Définition
ClearError	Efface la dernière erreur
GetLasterror	Récupère la dernière exception.
HtmlDecode	Transforme les caractères comme < en caractère (pour l'exemple : < conne <).
HtmlEncode	Remplace les caractères <, > et & par une chaine de caractère qui sera traduite par le serveur. C'est utile pour la sécurité. Par exemple si une personne essaye de rentrer du code avec du html qui sera plus tard afficher dans un label (comme dans son pseudo par exemple), et bien au lieu que le html soit exécuté il ne sera qu'affiché.
MapPath	Retourne le chemin physique du fichier spécifié en chemin (le fichier est spécifié depuis la racine de l'application).
Transfer	Arrête l'exécution de la page en cours et démarre celui de la page spécifié. L'url ne sera donc pas changé pour l'utilisateur qui vera l'url de la page d'origine même si le contenu est celui d'une autre page.
UrlDecode	Comme HtmlDecode sauf que c'est utilisé sur l'url.
UrlEncode	Comme HtmlEncode sauf que c'est utilisé sur l'url.
UrlPathEncode	Code la partie de l'url correspondant au chemin d'accès.

Voici un exemple utilisant la méthode HtmlEncode :

```
C#

protected void Page_Load(object sender, EventArgs e)
{
    Label Label1 = new Label();
    form1.Controls.Add(Label1);
    Label Label2 = new Label();
    form1.Controls.Add(Label2);

    string str = @"<h1> test </h1>";

    String myEncodedString;
    // On encode la chaine de caractères
    myEncodedString = Server.HtmlEncode(str);
    Label1.Text = "On affiche le string une fois qu'il a été encodé " +
myEncodedString + "<br />";
    StringWriter myWriter = new StringWriter();
    // On décode la chaine de caractères
    Server.HtmlDecode(myEncodedString, myWriter);
    Label2.Text = "Voici le string une fois décodé :" + myWriter.ToString();
}
```

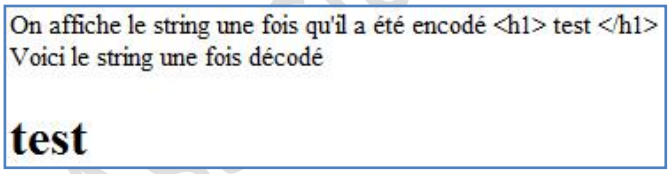
VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    Dim Label1 As Label = New Label()
    form1.Controls.Add(Label1)
    Dim Label2 As Label = New Label()
    form1.Controls.Add(Label2)

    Dim str As String = "<h1> test </h1>"

    Dim myEncodedString As String
    ' On encode la chaine de caractères
    myEncodedString = Server.HtmlEncode(str)
    Label1.Text = "On affiche le string une fois qu'il a été encodé " +
myEncodedString + "<br />"
    Dim myWriter As System.IO.StringWriter = New System.IO.StringWriter()
    ' On décode la chaine de caractères
    Server.HtmlDecode(myEncodedString, myWriter)
    Label2.Text = "Voici le string une fois décodé :" + myWriter.ToString()
End Sub
```

On obtient une page avec ceci dessus :



On remarque que le <h1> a été affiché en caractères et non pas interprété comme si c'était du code avec l'encodage. On remarquera d'ailleurs dans le code source de la page qu'il y a un bien un code qui représente les caractères < et > :

Code source de la page dans le navigateur

```
<span>
On affiche le string une fois qu'il a été encodé &lt;h1&gt; test &lt;/h1&gt;<br />
</span>

<span>Voici le string une fois décodé :<h1> test </h1></span>
```

3.1.4 Context

Méthodes	Définition
AddError	Ajoute une exception à la page.
ClearError	Efface la dernière erreur.

Propriétés	Définition
AllErrors	Récupère une collection d'exception non géré qui ont eu lieu sur la page.
IsCustomErrorEnabled	Obtient un booléen représentant le statut des erreurs personnalisés (à true elles sont activé)

IsDebuggingEnabled	Obtient un booléen qui est à true si le débogage est activé.
Timestamp	Récupère le timestamp auquel la requête HTTP à été créé.

3.2 Sur le navigateur

Il peut être utile de savoir quel navigateur est utilisé par vos visiteurs pour pouvoir leur proposer un site adapté. En effet tous les navigateurs n'ont pas le même affichage voir n'ont pas les mêmes fonctionnalités. Certain, par exemple, ne peuvent pas stocker de cookies, d'autres ne possèdent pas *l'ActiveX*. Si notre site en contient et qu'il n'est pas possible pour le navigateur de les prendre en compte, l'utilisation de votre site pourrait s'en voir réduite.

Savoir quel navigateur est utilisé est donc primordial pour pouvoir proposer aux utilisateurs, soit une alternative qui leur permettra d'avoir tout de même accès à la fonctionnalité, soit de faire apparaitre un message d'erreur pour l'informer qu'il ne peut pas y avoir accès et/ou lui recommandant d'utiliser un autre navigateur. Aussi, afin d'uniformiser votre site sur tous les navigateurs, faudra t-il le tester sur plusieurs navigateurs (au moins les plus répandu).

Pour récupérer des informations sur le navigateur de l'utilisateur, on va utiliser *Request.Browser*. Il contient plusieurs propriétés utiles pour connaitre le nom du navigateur, ainsi que ses fonctionnalités. Ci-dessous un tableau recensant les propriétés les plus utiles :

Propriétés	Définition
ActiveXControls	Renvoi un booléen indiquant si le navigateur peut utiliser ActiveX.
BackgroundSounds	Renvoi un booléen indiquant si le navigateur prend en charge le balise <bgsounds> qui permet de jouer des sons.
Browser	Obtient la chaine de caractères qui caractérise le nom du navigateur (exemple : Firefox, IE …). Il est possible que certain navigateur n'en ai pas ou que ce ne soit pas le bon. Nous le verrons un peu mieux plus tard.
ClrVersion	Obtient la valeur du Framework .Net installé sur le client.
Cookies	Renvoi un booléen indiquant si le navigateur prend en charge les cookies.
Crawler	Renvoi un booléen indiquant si le navigateur est un moteur de recherche.
Frames	Renvoi un booléen indiquant si le navigateur prend en charge les frames.
IsColor	Renvoi un booléen indiquant si le navigateur possède un affichage des couleurs.
IsMobileDevice	Renvoi un booléen indiquant si le navigateur est celui d'un téléphone reconnu.
JavaApplets	Indique si le navigateur peut prendre en charge du Java.
Javascript	Indique si le navigateur peut prendre en charge du Javascript.

Version	Obtient le numéro complet de la version du navigateur.

On peut par exemple utiliser l'évènement *Session_Start* qui comme nous l'avions vu, à partir d'un fichier *asax*, va nous permettre de définir avec du code behind ce que l'on veut faire au déclenchement d'une session. On va pouvoir à ce moment vérifier si le navigateur accepte les cookies, et si tel n'est pas le cas, utiliser un autre des moyens que nous avions vu (comme l'url où les champs caché) pour passer l'identifiant de session.

Voyons maintenant un exemple dans lequel nous allons récupérer le nom et la version du navigateur pour l'afficher :

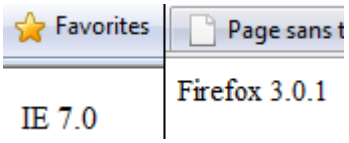
C#

```
protected void Page_Load(object sender, EventArgs e)
{
    Label Label1 = new Label();
    Label1.Text = Request.Browser.Browser;
    Label1.Text += " ";
    Label1.Text += Request.Browser.Version;
    form1.Controls.Add(Label1);
}
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    Dim Label1 As Label = New Label()
    Label1.Text = Request.Browser.Browser
    Label1.Text += " "
    Label1.Text += Request.Browser.Version
    form1.Controls.Add(Label1)
End Sub
```

Voici, ci-contre, ce que vous pouvez obtenir avec Internet Explorer et Mozilla Firefox. On aurait aussi bien pu changer



3.3 Les Headers

Les headers sont des informations qui sont stockées dans l'en-tête de la requête *HTTP*. *ASP.NET* nous permet de les lire et de les modifier.

Il y a deux objets qui permettent d'accéder aux headers : l'objet *Page* et l'objet *Response*. L'objet *Response* permet de modifier les headers dans le flux de réponse qui va être envoyé à l'utilisateur.

Voici le tableau des propriétés de Page.Header. Il permet de récupérer et de modifier les headers déjà existant. Il ne permet pas d'en ajoutant ou d'en supprimer.

Propriétés	Définition
StyleSheet	Permet de modifier le style qui sera utilisé.
Title	Permet de modifier le titre de la page.

Plus exactement les headers sont les informations qui sont contenues dans la balise *head*. Aussi pour que l'on puisse modifier ou lire ces informations depuis le *code b*ehind, il faut que cette balise possède la propriété *runat="server"*.

Pour exemple voici comment changer le titre d'une page depuis le *code b*ehind :

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    Page.Header.Title = "Test";
}
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    Page.Header.Title = "Test"
End Sub
```

4 Conclusion

A la suite de ce chapitre, nous sommes capable de récupérer et/ou de modifier des informations relatives à l'application ou au client comme le navigateur utilisé, ses possibilités, les headers qu'on envoi au client ou ceux qu'il a envoyé ….

En fonction de l'endroit où l'on se trouve dans le code on n'a pas accès aux mêmes objets, mais il est possible d'accéder de différentes façons à la même information. Ainsi on peut retrouver la même méthode sur plusieurs objets. Le prochain chapitre va porter sur l'affichage et les thèmes, on y verra comment gérer ce qui concerne l'affichage et le système des thèmes (avec skin) fourni par ASP.NET.