Université Libanaise
Faculté de Technologie
Génie des Réseaux Informatiques et Télécommunications

الجامعة اللبنانيّة
كلية التكنولوجيا
قسم هندسة شبكات المعلوماتية والاتصالات

# Architecture Client Serveur

## Le Langage ASP.NET – Traitement des événements et Gestion des états


*Préparé par:*
*Dr. Youssef ROUMIEH*
*E-mail :* youssef.roumieh@gmail.com

*Reference:* **Web Applications Development with Microsoft .NET Framework 4.**
 ISBN: 978-0-7356-2740-6

*Les problèmes rencontrés et les oublis constatés dans ce support sont à signaler à* youssef.roumieh@gmail.com

- Applications Windows et Applications Web
  - La différence n'est pas l'interface utilisateur, c'est le cycle de vie.

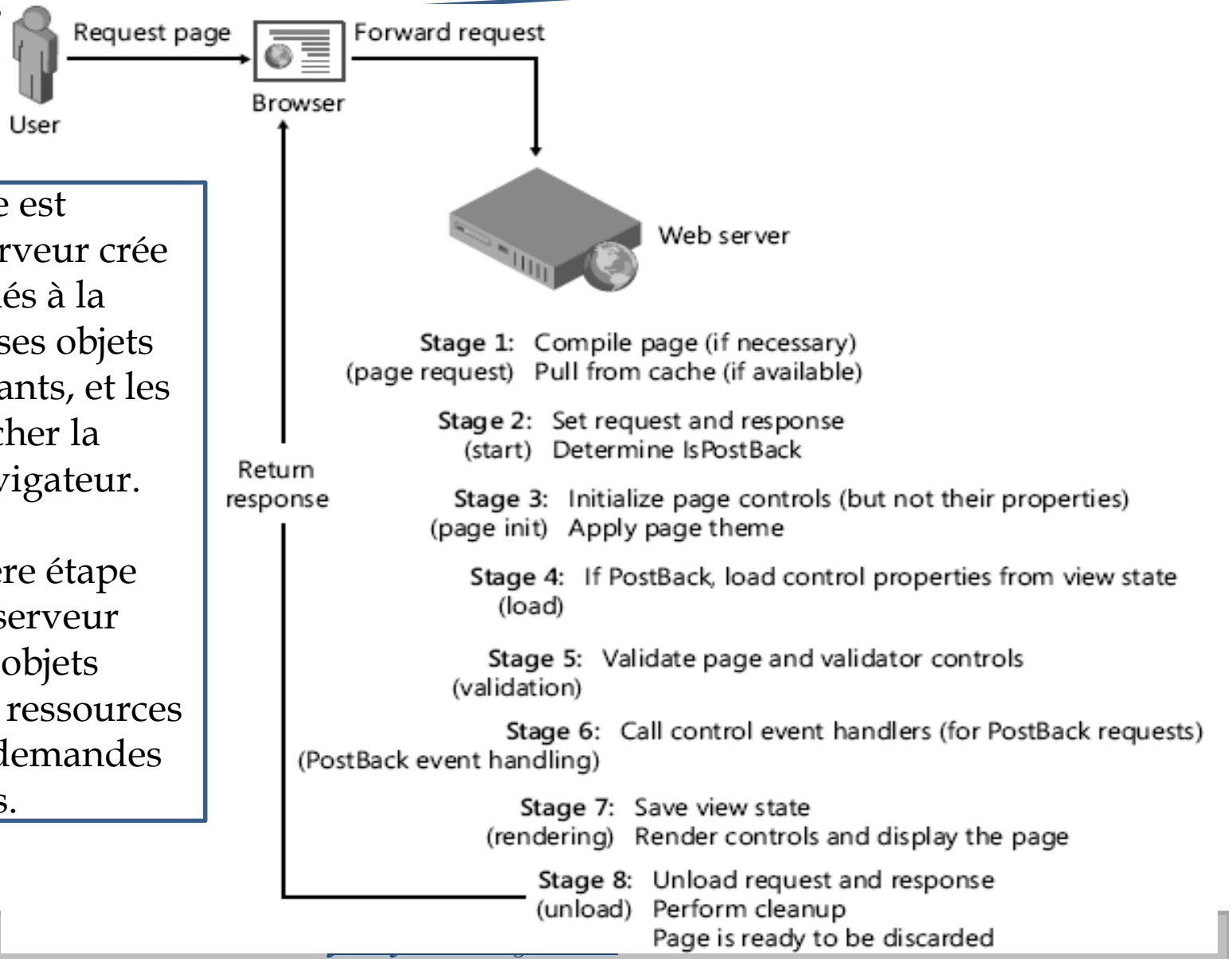
- Chaque requête web individuelle **nécessite la création des contrôles et des objets et finalement la destruction d'eux**.
  - **Les objets que vous créerez ne plus être disponibles lors de la prochaine demande de l'utilisateur**.

- Les applications ASP.NET s'exécutent sur un serveur web. **Le serveur Web est généralement Internet Information Services (IIS).**

# Comprendre le cycle de vie d'une page Web d'ASP.NET et ses contrôles

Quand une page est demandée, le serveur crée des objets associés à la page, avec tous ses objets de contrôles enfants, et les utilise pour afficher la page dans le navigateur.

Quand la dernière étape est terminée, le serveur Web détruit ces objets pour libérer des ressources pour traiter les demandes supplémentaires.



# Evénements au niveau de l'application

- **Application_Start:**
  - déclenché lorsque votre application est lancée par IIS.
  - Cet événement est utile pour l'**initialisation de variables** qui ont une portée au **niveau de l'application**.
- **Application_End:**
  - déclenché lorsque votre application s'arrête ou s'éteint.
  - Cet événement est utile si vous avez besoin de **libérer des ressources** au niveau des applications ou effectuer une **sorte de journalisation**.
- **Application_Error:**
  - déclenché quand une erreur non traitée se produit et monte à la portée de l'application.
  - Vous pouvez utiliser cet événement pour effectuer le pire des cas, **capture tout journalisation des erreurs**.
- **Application_LogRequest:**
  - déclenché lorsqu'une **demande a été faite à l'application**.
  - Vous pouvez utiliser cet événement pour écrire les informations de **journalisation** personnalisée concernant une demande.

# Le fichier Global.asax (Classe d'application globale) :

- Vous pouvez implémenter les gestionnaires des **événements d'application** en ajoutant un fichier **Global.asax** à votre projet.
- Vous pouvez ajouter le fichier Global.asax (s'il n'est pas créé) : Add New Item → Global Application →Add
- Exemple: La variable du niveau application **UsersOnline** est définie au démarrage de l'application.

```
<%@ Application Language="C#" %>
<script runat="server">
void Application_Start(object sender, EventArgs e){
      Application["UsersOnline"] = 0;
}
void Session_Start(object sender, EventArgs e){
      Application.Lock();
      Application["UsersOnline"] =
            (int)Application["UsersOnline"] + 1;
      Application.UnLock();
}
void Session_End(object sender, EventArgs e){
      Application.Lock();
      Application["UsersOnline"] =
            (int)Application["UsersOnline"] - 1;
      Application.UnLock();
}
</script>
```

Because multiple web-pages might be running simultaneously on different threads, you must lock the Application object.

The values of an Application variable are of the Object type → you must cast them to the appropriate type.

# Evénements du cycle de vie d'une page

| Event | Description |
|---|---|
| **PreInit** | Le premier événement réel traité pour une page. Généralement, cet événement est utilisé uniquement si vous avez besoin de définir des valeurs telles que page maître ou thème. |
| **Init** | Cet événement se déclenche après que chaque contrôle a été initialisé. Vous pouvez utiliser cet événement pour modifier les valeurs d'initialisation des contrôles. |
| **Load** | La page est stable à ce moment, il a été initialisé et son état a été reconstruit. Le code dans l'événement de chargement de la page (page load) vérifie habituellement pour postback (n'est pas la première demande de la page), puis définit les propriétés de contrôle appropriée.<br>L'événement de chargement de la page est appelé en premier. Ensuite, l'événement de chargement pour chaque contrôle enfant est appelé à son tour (et puis les événements de chargement pour leurs contrôles enfants, le cas échéant). |
| **Control (postback) event(s)** | ASP.NET appelle maintenant tous les événements de la page ou de ses contrôles qui a provoqué la publication (postback). C'est peut-être l'événement de clic sur un bouton, par exemple. |
| **PreRender** | Cet événement permet les modifications finales à la page ou à son contrôle. Elle a lieu après que tous les événements de publication régulière ont eu lieu. Cet événement a lieu avant que ViewState est enregistrée, donc les modifications apportées ici sont sauvegardées. |

# Création de contrôles personnalisés

- Vous pouvez ajouter des contrôles à un formulaire au moment de l'exécution de la page.
  - Événement *Page.PreInit* (si vous n'utilisez pas master page).
  - Evénement *Page.Init* (si vous utilisez master page).

- **Exemple :**
  - Supposons qu'on a ajouté un contrôle de type **Panel** (panel1) à votre page ASPX à l'endroit dans lequel vous souhaitez les contrôles apparaissent.
  - Vous pouvez également utiliser un contrôle **PlaceHolder**.

```
protected void Page_PreInit(object sender, EventArgs e){
        // Create instances of the controls
        Label FeedbackLabel = new Label();
        TextBox InputTextBox = new TextBox();
        Button SubmitButton = new Button();

    // Assign the control properties
    //la propriété ID vous permet d’accéder aux contrôles à partir
    // d’autres méthodes.
        FeedbackLabel.ID = "FeedbackLabel";
        FeedbackLabel.Text = "Please type your name: ";
        SubmitButton.ID = "SubmitButton";
        SubmitButton.Text = "Submit";
        InputTextBox.ID = "InputTextBox";

        // Create event handlers
        SubmitButton.Click += new
        System.EventHandler(SubmitButton_Click);

        // Add the controls to a Panel
        Panel1.Controls.Add(FeedbackLabel);
        Panel1.Controls.Add(InputTextBox);
        Panel1.Controls.Add(SubmitButton);
}
```

```
protected void SubmitButton_Click(object sender, EventArgs e)
{
    // Create an instance of Button for the existing control
    Button SubmitButton = (Button)sender;

    // Update the text on the Button
    SubmitButton.Text = "Submit again!";

    // Create the Label and TextBox controls
    Label FeedbackLabel = (Label)FindControl("FeedbackLabel");
    TextBox InputTextBox = (TextBox)FindControl("InputTextBox");

    // Update the controls
    FeedbackLabel.Text = string.Format("Hi, {0}", InputTextBox.Text);
}
```

- **Remarque :**
  - **Les instances que vous créez dans Page_PreInit ou Page_Init ne seront pas disponibles dans les gestionnaires d'événements.** Par conséquent, vous aurez besoin de trouver les contrôles et les cast au bon type (FindControl).

# Création de gestionnaires d'événements

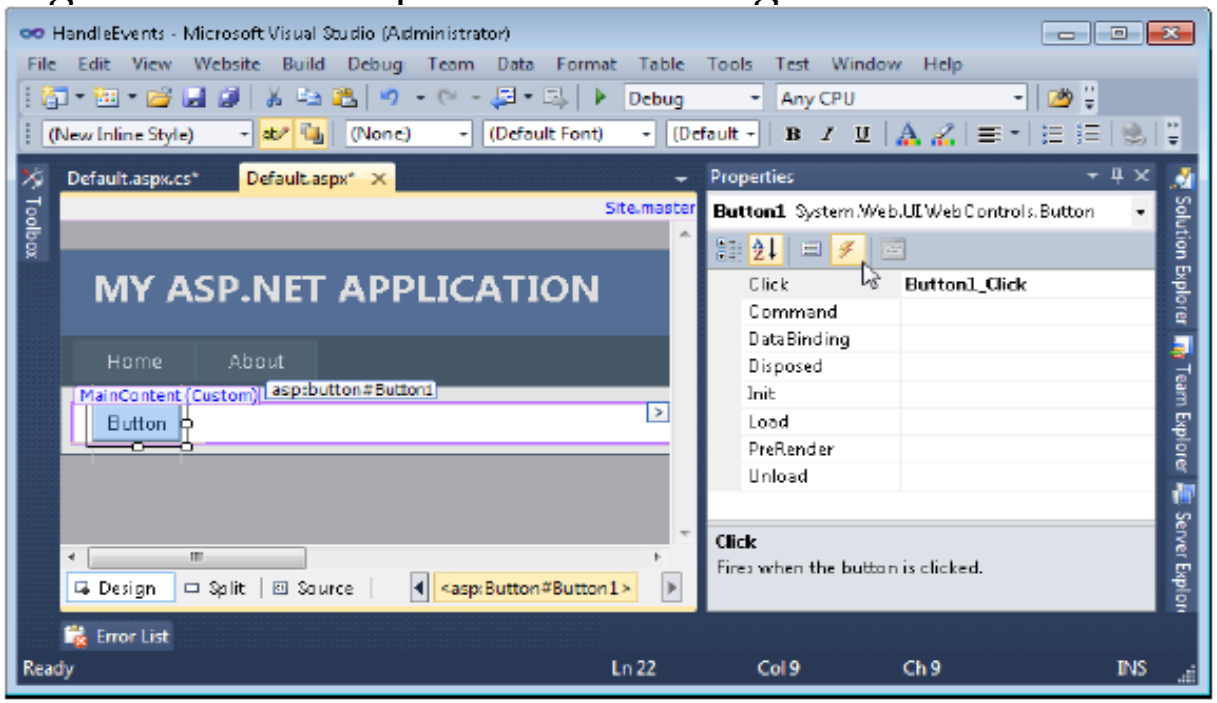
- Les contrôles dans ASP.NET ont des **événements par défaut.** Par exemple**, l'événement par défaut de l'objet Page est l'événement Load, et l'événement par défaut de l'objet Bouton est l'événement Click**.
  - Design View → **double-click** sur le contrôle considéré
    - Visual Studio crée l'événement dans le fichier code-behind.

- Pour ajouter un événement autre que l'événement par défaut, par exemple ajouter le gestionnaire d'événement Init à une page (Page_Init), suivre les étapes suivantes :
  - Solution Explorer → Right-click sur la page Web →sélectionner **View Markup** pour ouvrir le code de conception (HTML). **Vérifiez que la propriété AutoEventWireup de la page est définie à true dans la directive @ Page (valeur par défaut)**. Cela signifie que le gestionnaire d'événement dans le code a le format Page_EventName .
  - Ensuite, Right-click sur la même page et cliquer sur **View Code** pour ouvrir l'éditeur de code avec **le fichier code-behind**. Ajouter le gestionnaire d'événement :

```
private void Page_Init(object sender, EventArgs e)
{       }
```

# Ajouter un gestionnaire d'événements à un contrôle serveur C#

- Ouvrez une page en Design View. Ajoutez un contrôle Button à partir de Toolbox.
- Right-click sur le contrôle Button et sélectionner Properties.
- Dans la fenêtre Properties, cliquez sur l'icône des événements (l'éclair jaune). Cela change la fenêtre Propriétés à l'affichage des événements (events view).




- Double-click sur l'événement que vous souhaitez traiter. Cela va ouvrir la page code-behind avec le gestionnaire de l'événement.

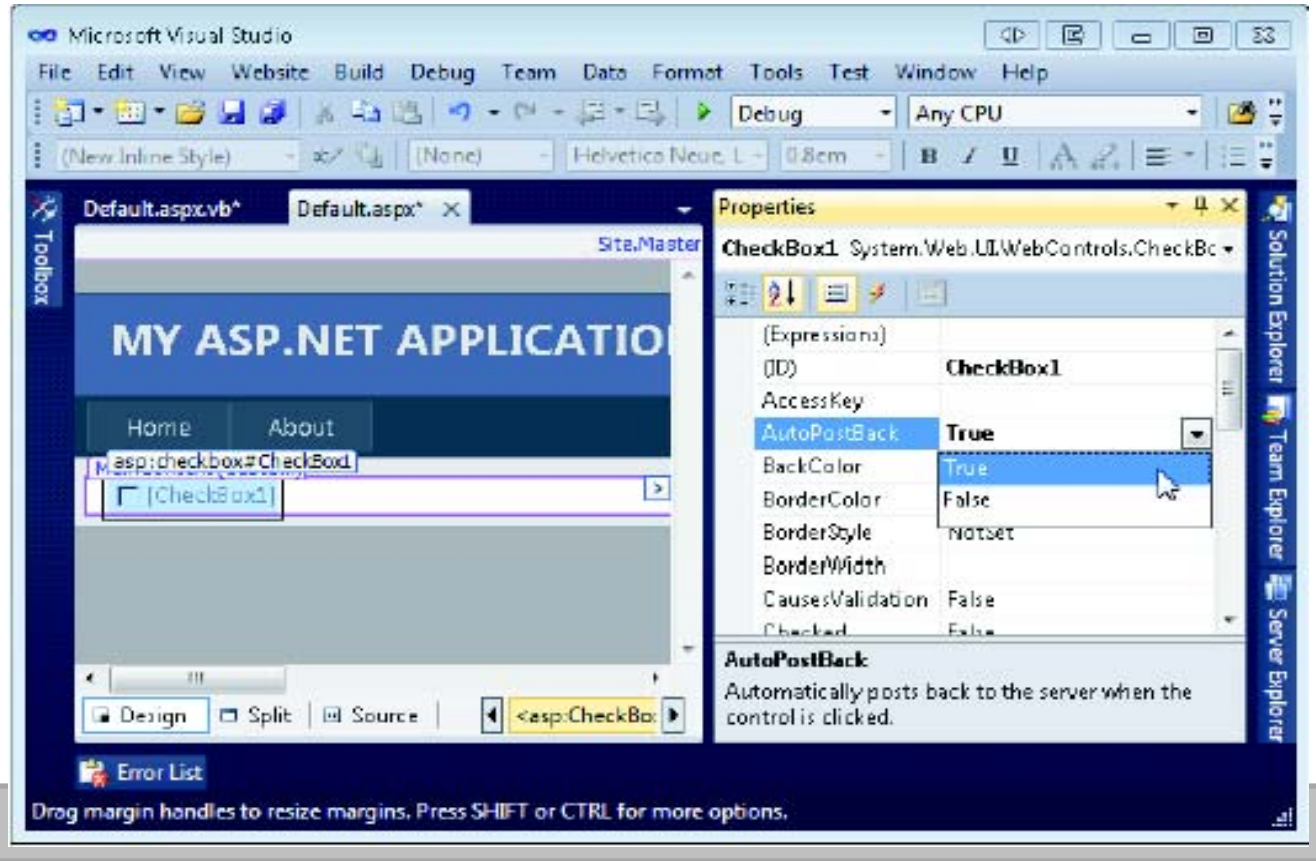
# Contrôle automatique Postback

- Certains contrôles provoquent une soumission de la page (postback) lorsqu'un événement particulier se produit.
  - Par exemple, le navigateur soumet le formulaire au serveur web lorsqu'un utilisateur clique sur un contrôle de type Button.

- Si un utilisateur sélectionne un élément d'une DropDownList, ce dernier ne soumet pas au serveur. Ainsi, la zone de texte (TextBox) contient un événement par défaut appelé TextChanged. Cet événement ne provoque pas une soumission automatique, rien ne se passe sur le serveur au moment où l'utilisateur change le texte.
  - **L'événement n'est pas perdu, ASP.NET déclenche l'événement lors de la prochaine soumission, qui se produit généralement lorsque l'utilisateur clique sur un bouton.**
  - Tout événement reporté (un événement déclenché par un utilisateur qui ne provoque pas une soumission automatique) s'exécute avant l'événement réel qui a provoqué la soumission.

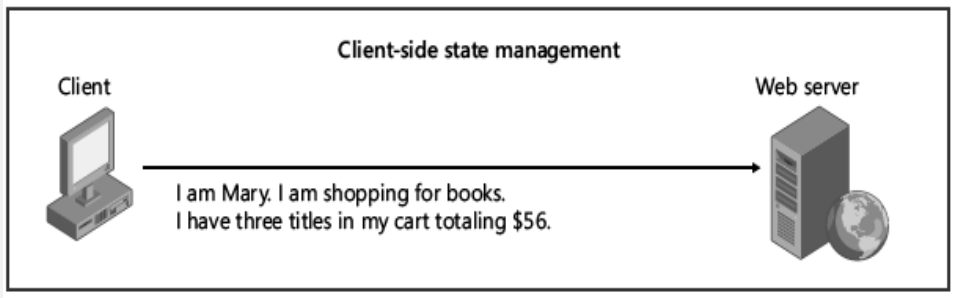
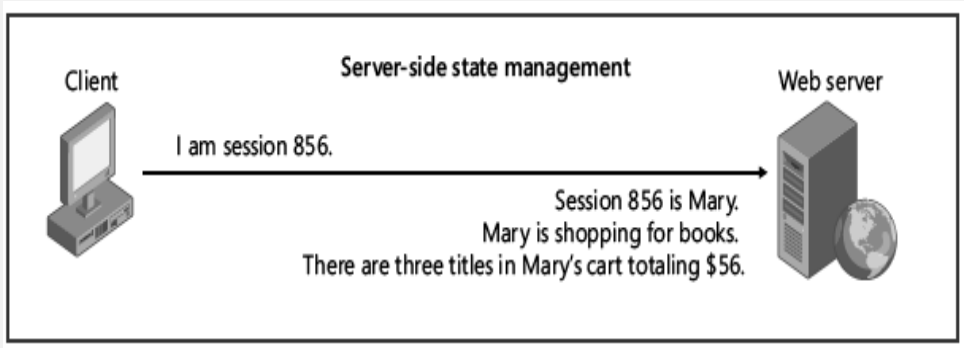
# Définir AutoPostBack à la valeur true

- La propriété **AutoPostBack pour un contrôle est utilisée pour indiquer si l'événement par défaut de ce contrôle provoque une publication automatique sur le serveur.**
  - Attribuer AutoPostBack à la valeur true dans la fenêtre des propriétés (figure)
- Ajouter l'attribut AutoPostBack = "True" au contrôle en mode Source.



# Choisir entre la gestion d'état côté client ou celle côté serveur

- Les informations telles que le nom d'utilisateur, options de personnalisation, ou contenu de caddy, peut être stockée sur le client ou sur le serveur.

| | |
|---|---|
| **Gestion de l'état – cote Client :**<br><br>**le client envoie les informations au serveur à chaque requête.** |  |
| **Gestion de l'état – cote Serveur :**<br><br>**Le serveur stocke les informations, mais le serveur suit le client à l'aide d'une technique de gestion de l'état – côté client.** |  |

- **Avantages de stocker les informations chez le client**
  - **une meilleure évolutivité (better scalability):**
    - moins de **consommation de la mémoire du serveur**. Servir plus de demandes.
  - **Support pour plusieurs serveurs web:**
    - Fournir les informations à n'importe quel serveur web.
- **Avantages de stocker les informations sur le serveur**
  - **une meilleure sécurité:**
    - **Les utilisateurs peuvent modifier les informations** de gestion d'état côté client. Vous ne devez jamais utiliser la gestion d'état côté client pour stocker des informations confidentielles telles que mots de passe, les niveaux d'accès, ou l'état d'authentification.
  - **Bande passante réduite:**
    - La sauvegarde de grandes quantités d'information sur le client augmente l'utilisation de la bande passante et les temps de chargement des pages, puisqu'à chaque aller-retour les informations sont envoyées.

- Le choix que vous faites pour gérer l'état de l'application doit être décidé en fonction de ces compromis. En général, on utilise un mélange de la gestion d'état de côté client et côté serveur.

# Utilisation de la gestion de l'état côté client

- Si votre application a besoin de s'adapter à des **milliers d'utilisateurs**, alors vous devriez sérieusement envisager **d'utiliser le client pour stocker l'état d'application**.

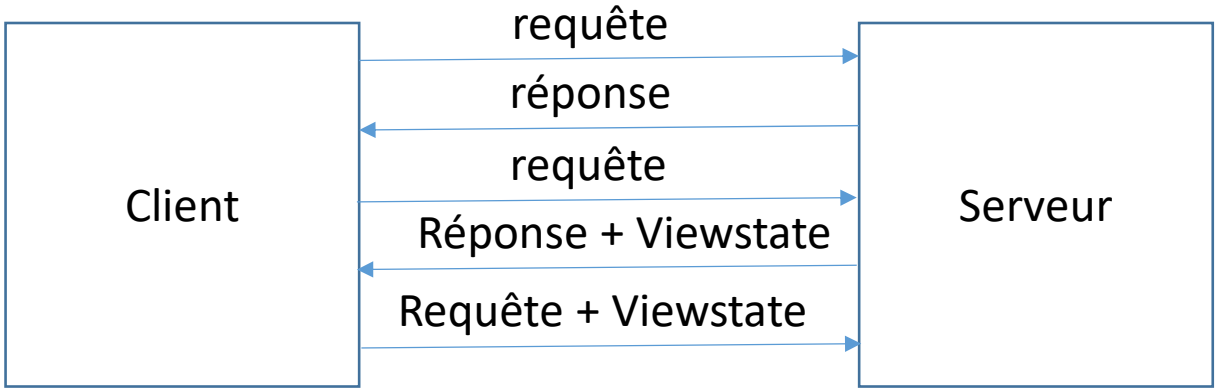
- **La suppression de cette charge du serveur libère des ressources, permettant au serveur de traiter plus de demandes des utilisateurs**.

- ASP.NET fournit plusieurs techniques pour stocker des informations d'état sur le client.

# État de l'affichage (View State)

- ASP.NET utilise l'état d'affichage pour garder les valeurs dans les contrôles entre les demandes de page.
  - Par exemple, si un utilisateur tape une adresse dans une zone de texte et l'état d'affichage est activé, l'adresse restera dans la zone de texte entre les requêtes.
- Vous pouvez également ajouter vos propres valeurs personnalisées à l'état d'affichage.
- Lorsque l'utilisateur effectue la requête suivante, l'état d'affichage est retourné avec sa demande.

- ASP.NET tire l'état d'affichage de la page et l'utilise pour réinitialiser les valeurs des propriétés de la page.



- La propriété Page.ViewState fournit un objet de dictionnaire pour conserver les valeurs entre plusieurs demandes de la même page. Cet objet est du type StateBag.

- Quand une page ASP.NET est traitée, l'état actuel de la page et de ses contrôles est haché dans une chaîne et enregistré dans la page HTML dans un champ caché appelé __ ViewState (**plus de sécurité que les champs cachés simples de HTML)**.

- Exemple:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE“
value="/wEPDwULLTEzNjkxMzkwNjRkZAVvqsMGC6PVDmbCxBlPkLVKNahk" />
```

# Considérations sur la sécurité de l'État d'affichage

- L'**état d'affichage peut être altéré** puisqu'elle est un **champ caché dans le navigateur de l'utilisateur**.
  - Vous pouvez compter sur le fait que **l'état d'affichage est haché** et codé avant d'être envoyé au navigateur de l'utilisateur.
  - L'état d'affichage comprend également **un code d'authentification de message** (MAC: message authentication code). Ce MAC est utilisé par ASP.NET pour déterminer si l'état d'affichage a été altéré.
  - Cela permet de **garantir la sécurité dans la plupart des situations**, **sans exiger un état d'affichage entièrement crypté**.

- Si vous avez des informations très sensibles qui sont stockées dans l'état d'affichage entre les demandes de page, vous pouvez **chiffrer l'état d'affichage en utilisant la propriété ViewStateEncryptionMode de l'objet Page**.
  - Cela **assurera l'état d'affichage** mais également **diminuera le rendement global** de la transformation de la page, en raison du cryptage et de décryptage de données.
  - Il **augmentera la taille des données** envoyées transmises entre le navigateur et le serveur.

- Pour **activer le mode de cryptage de l'Etat d'affichage pour tout le site**, il suffit d'attribuer **Always** à **l'attribut viewStateEncryptionMode** de l'élément **pages** dans le fichier **web.config.**

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

- Alternativement, vous pouvez contrôler le **chiffrement de l'état d'affichage au niveau de la page**.

```
<%@ Page Language="C#" AutoEventWireup="true"
ViewStateEncryptionMode="Always"%>
```

- **Remarque :**
  - Parce que l'état d'affichage prend en charge le chiffrement, il est considéré comme la méthode la plus sécurisée de la gestion des états côté client.
- Cryptée l'état d'affichage est assez sûr pour la plupart des exigences de sécurité, mais il est plus sûr de stocker les données sensibles sur le serveur et ne pas les envoyer au client.

# Désactivation les données de l'Etat d'affichage

- **L'état d'affichage est activé (enabled) par défaut** pour votre page et tous les contrôles de la page.

- Parce que les données d'état d'affichage sont envoyées en aller/retour entre le navigateur et le serveur, vous devez toujours **minimiser la quantité de données de l'état d'affichage pour améliorer les performances**.
  - **Désactiver l'état d'affichage** des contrôles et des pages entières si c'est possible.
  - **Au niveau de la page**, vous pouvez contrôler l'état d'affichage en activant et désactivant **Page.EnableViewState** et **Page.ViewStateMode**.
  - **Pour les contrôles**, utiliser **Control.EnableViewState** et **Control.ViewStateMode**.

# Les paramètres de l'Etat d'affichage

| | PAGE. ENAbLE VIEWSTATE | PAGE. VIEWSTATE MODE | CONTROL. ENAbLE VIEWSTATE | CONTROL. VIEWSTATEMODE |
|---|---|---|---|---|
| Disable view state for a page, but enable it for a specific control | True | Disabled | True | Enabled |
| Disable view state for a page and disable it for a specific control | True | Disabled | False | Disabled or Inherit |
| Enable view state for a page, but disable it for a specific control | True | Enabled | False | Disabled |
| Enable view state for a page and for a control | True | Enabled | True | Enabled or Inherit |

- Vous pouvez également désactiver l'état d'affichage dans un fichier Web.config, en ajoutant l'entrée de configuration suivante à la section <system.web> :

```
<configuration>
<system.web>
<pages enableViewState="false">
</system.web>
</configuration>
```

# Lecture et écriture des données personnalisées dans l'État de l'affichage

- Vous pouvez utiliser l'état d'affichage pour ajouter et pour récupérer des valeurs personnalisées (**Ces valeurs ne sont pas des parties d'un contrôle**) que vous avez besoin de persister entre les demandes de page**.**
- Exemple :

**Sample of Visual Basic Code**

```
'writing to view state
Me.ViewState.Add("MyData", "some data value")
'read from view state
Dim myData As String = CType(ViewState("MyData"), String)
```

**Sample of C# Code**

```
//writing to view state
this.ViewState.Add("MyData", "some data value");
//read from view state
string myData = (string)ViewState["MyData"];
```

- Ajouter des données à l'état d'affichage lorsque vous avez besoin de transmettre des informations au serveur dans le cadre de la publication de la page.
  - Le contenu de l'état d'affichage est pour cette page seulement.
  - **L'état d'affichage ne transfère pas d'une page Web à une autre (comme un cookie).**

- **Exemple :** Stockage d'une instance de l'objet DateTime dans le ViewState sans le convertir en une chaîne.

**Sample of Visual Basic Code**

```
' Check if ViewState object exists, and display it if it does
If (Me.ViewState("lastVisit") IsNot Nothing) Then
   Dim lastVisit As DateTime = CType(Me.ViewState("lastVisit"),
DateTime)
   Label1.Text = lastVisit.ToString()
Else
  Label1.Text = "lastVisit ViewState not defined!"
End If
' Define the ViewState object for the next page view
Me.ViewState("lastVisit") = DateTime.Now
```

**Sample of C# Code**

```
// Check if ViewState object exists, and display it if it does
if (ViewState["lastVisit"] != null)
   Label1.Text = ((DateTime)ViewState["lastVisit"]).ToString();
else
   Label1.Text = "lastVisit ViewState not defined.";
// Define the ViewState object for the next page view
ViewState["lastVisit"] = DateTime.Now;
```

# État du contrôle (Control State)

- vous permet de conserver les informations sur un contrôle qui ne fait pas partie de l'état d'affichage.

- **Ceci est utile pour les développeurs de contrôles personnalisés.** Si l'état d'affichage est désactivé pour un contrôle ou la page, l'état de contrôle continue à fonctionner.

# Champs cachés (hidden fields)

- Un champs caché n'est pas affiché à l'utilisateur (sauf si l'utilisateur choisit de voir la source de la page), puis envoyés vers le serveur lorsque la page est publiée (POST).
- Le **contrôle HiddenField** vous permet de stocker des données dans sa **propriété Value**.
- Comme l'état d'affichage, les champs cachés **ne stockent des informations que pour une seule page**.
- Contrairement à l'état d'affichage, les champs cachés **n'ont pas intégré la compression, le cryptage, le hachage ou l'arrachement**. Par conséquent, les utilisateurs peuvent afficher ou modifier les données stockées dans des champs cachés.
- **Pour utiliser les champs cachés, vous devez soumettre vos pages au serveur en utilisant HTTP POST** (ce qui arrive en réponse à un utilisateur appuyant sur un bouton de soumission).
- Vous ne pouvez pas simplement appeler un HTTP GET (ce qui arrive si l'utilisateur clique sur un lien) et récupérer les données du champ caché sur le serveur.

- **Remarque :** La technique de l'Etat d'affichage et les champs cachés **ne fonctionnent que pour des demandes multiples sur une seule page**.

# Cookies

- Un cookie est une **petite quantité de données stockée chez le client** et transmise avec les demandes de votre site.
  - Le navigateur envoie cette donnée avec chaque requête d'une page au même serveur.

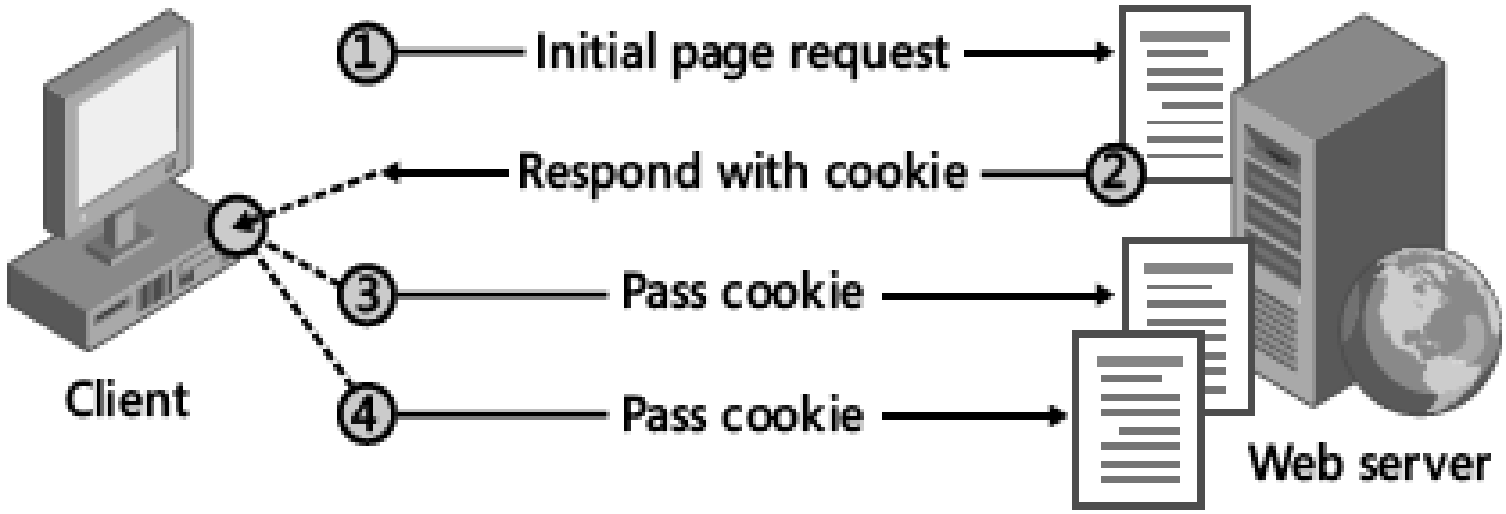
- Types de cookies :
  - **Cookies Persistant :** Les cookies sont **des fichiers texte** sur l'ordinateur client. Ces cookies restent existants même si l'utilisateur ferme le navigateur et rouvre à un moment ultérieur.
  - **Cookies Temporaires :** stockés dans la mémoire du navigateur du client.
    - Ces cookies ne sont utilisés que lors de la session Web en cours.
    - Ils sont perdus lorsque le navigateur se ferme.

# Cookies

- **Les cookies sont la meilleure façon de stocker les données d'état qui doivent être disponibles pour de multiples pages Web sur un site Web**.
  - Les applications Web ont souvent **besoin de suivre les utilisateurs entre les demandes de page**.
  - Ces applications doivent assurer que l'utilisateur effectue la première requête est le même utilisateur effectuant les demandes subséquentes.

- **Les cookies peuvent être consultés à travers des visites différentes (par exemple, les préférences de l'utilisateur)**.

# Les serveurs Web utilisent des cookies pour suivre les clients Web.




- Les **utilisateurs peuvent supprimer les cookies** de leur ordinateur à tout moment.
- Les cookies **ne permettent pas de résoudre le problème d'un utilisateur déplaçant d'un ordinateur à un autre** (ou d'un ordinateur vers un appareil mobile).
  - Dans ces cas, les préférences des utilisateurs ne vont pas toujours avec eux.
  - L'utilisateur besoin de se connecter pour réinitialiser leurs cookies.

# Lecture et écriture des Cookies

- Une application web crée un cookie par **l'envoi au client comme un en-tête dans une réponse HTTP**.

- Pour **ajouter un cookie (**instances de HttpCookie) à la collection de cookies, vous **appelez la méthode Response.Cookies.Add**.
  - L'objet de **HttpCookie** contient simplement une propriété **Name** et une propriété **Value**.
  - Exemple :

- Response.Cookies.Add(New HttpCookie("userId", userId))

- Pour **récupérer un cookie** renvoyé par le navigateur Web, les valeurs sont dans la collection **Request.Cookies.**
  - Exemple : Request.Cookies("userId").Value

# Exemple

Sample of C# Code

```
// Check if cookie exists, and display it if it does
if (Request.Cookies["lastVisit"] != null)
   // Encode the cookie in case the cookie contains client-side script
   Label1.Text = Server.HtmlEncode(Request.Cookies["lastVisit"].Value);
else
   Label1.Text = "No value defined";

// Define the cookie for the next visit
Response.Cookies["lastVisit"].Value = DateTime.Now.ToString();
Response.Cookies["lastVisit"].Expires = DateTime.Now.AddDays(1);
```

- **Remarque**
  - La propriété Expires détermine la période de temps que le cookie sera stocké chez le client. Ceci permet de persister le cookie entre les sessions de navigateur.
    - **Si vous ne définissez pas la propriété Expires, le navigateur stocke le cookie en sa mémoire et le cookie sera perdu lorsque l'utilisateur ferme son navigateur.**
  - **Pour supprimer un cookie, vous écrasez le cookie en définissant une date d'expiration dans le passé. Vous ne pouvez pas supprimer directement les cookies car ils sont stockés sur l'ordinateur du client**.

# Contrôler la portée du Cookie

- Les informations contenues dans **les cookies sont généralement spécifiques à un site et sont souvent privées**. Pour cette raison, un navigateur ne doit pas envoyer votre cookie à un autre site.
- **La portée de votre cookie détermine les pages qui ont accès à l'information intégrée dans le cookie.**
  - Si vous limitez la portée à un répertoire, seules les pages de ce répertoire auront accès à l'information du cookie.
  - **Pour limiter le champ d'application d'un cookie** à un répertoire, vous définissez la propriété **Path** de la classe **HttpCookie**.

```
Sample of C# Code
Response.Cookies["lastVisit"].Value = DateTime.Now.ToString();
Response.Cookies["lastVisit"].Expires = DateTime.Now.AddDays(1);
Response.Cookies["lastVisit"].Path = "/MyApplication";
```

  - Pour étendre la portée d'un cookie à un domaine entier, définissez la propriété **Domain** de la classe **HttpCookie**. Le code suivant illustre cela.

```
Response.Cookies["lastVisit"].Value = DateTime.Now.ToString();
Response.Cookies["lastVisit"].Expires = DateTime.Now.AddDays(1);
Response.Cookies["lastVisit"].Domain = "contoso.com";
```

# Stocker plusieurs valeurs dans un cookie

- La taille de votre cookie est **dépendante du navigateur**. Chaque cookie peut contenir **jusqu'à 4 Ko** de données.
- Vous pouvez généralement **stocker jusqu'à 20 cookies par site**. Cela devrait être plus que suffisante pour la plupart des sites.
- **Vous pouvez stocker plusieurs valeurs dans un seul cookie en définissant le nom du cookie et sa valeur de clé.**
  - La valeur de la clé n'est généralement pas utilisée pour stocker une seule valeur.
  - Toutefois, si vous avez besoin de plusieurs valeurs dans un seul cookie, vous pouvez ajouter plusieurs clés.
  - Exemple :

```
Response.Cookies["info"]["visit"] = DateTime.Now.ToString();
Response.Cookies["info"]["firstName"] = "Tony";
Response.Cookies["info"]["border"] = "blue";
Response.Cookies["info"].Expires = DateTime.Now.AddDays(1);
```

    - L'exécution de ce code envoie un seule cookie au navigateur Web. Les valeurs envoyées sont : (visit=4/5/2006 2:35:18 PM) (firstName=Tony) (border=blue)
    - **Les propriétés des cookies, comme Expires, Domain and Path, s'appliquent pour toutes les valeurs dans un seul cookie.**
    - Vous pouvez accéder aux valeurs individuelles d'un cookie par l'intermédiaire Request.Cookies de la même façon dont vous définissez les valeurs (utilisant à la fois le nom et la clé).

# Chaînes de requête (Query Strings)

- Les chaînes de requête annexées aux URLs sont couramment utilisées pour stocker les valeurs des variables qui identifient le contexte spécifique d'une page demandée.
  - Ce contexte pourrait être un terme de recherche, numéro de page, l'indicateur de région, ou quelque chose semblable.
- Les chaînes de requête sont mis en valeur à partir de l'URL avec un point d'interrogation (?), Suivi par le nom de paramètre, un signe égal (=), et la valeur.
  - Vous pouvez ajouter plusieurs paramètres de chaîne de requête en utilisant l'ampersand (&).
- Exemple :
  - http://support.microsoft.com/Default.aspx?kbid=315233
  - http://search.microsoft.com/results.aspx?mkt=en-US&setlang=en-US&q=hello+world

# Accès aux paramètres de la chaîne de requête

- Valeurs envoyées à votre page via la chaîne de requête peuvent être récupérées par le serveur via la propriété **Page.Request.QueryString**.
- Exemple :
  - http://search.microsoft.com/results.aspx?mkt=en-US&setlang=en-US&q=hello+world

| Parameter Name | ASP.NET Object call | Value |
|---|---|---|
| mkt | Request.QueryString["mkt"] | en-US |
| setlang | Request.QueryString["setlang"] | en-US |
| Q | Request.QueryString["q"] | hello world |

- **Ajoutant les paramètres de chaîne de requête à une URL**
  - Vous devez ajouter manuellement les valeurs de chaîne de requête pour chaque lien hypertexte que l'utilisateur peut cliquer.
    - Par exemple, si vous avez un contrôle HyperLink avec NavigateUrl définie comme page.aspx, vous pouvez ajouter la chaîne « ?user=mary » à la propriété HyperLink.NavigateUrl de sorte que l'URL complète est page.aspx?user=mary.

- **Limitations :**
  - Les chaînes de requête constituent un moyen simple mais limité pour conserver les informations d'état entre plusieurs pages.
  - Certains navigateurs imposent une limite de 2083 caractères sur la longueur de l'URL.
  - Vous devez soumettre la page en utilisant une commande HTTP GET de sorte que les valeurs de chaîne de requête seront disponibles pendant le traitement.
  - Les utilisateurs peuvent facilement modifier les chaînes de requête (les paramètres et les valeurs sont visibles pour l'utilisateur dans sa barre d'adresse).
    - **V**ous **devez toujours valider les données extraites à partir d'une chaîne de requête**.

- **Avantages**
  - Si vous avez besoin de stocker des informations sur une page de sorte que **le lien peut être sauvegardé ou partagé**, ajoutez-le à la chaîne de requête : les signets (*bookmarks),* envoi par e-mail.
    - les chaînes de requête sont le seul moyen permettant à un utilisateur d'inclure les données d'état lorsque vous copiez et colliez une URL à un autre utilisateur.

# Lecture des paramètres de chaîne de requête dans votre page

- Pour lire une valeur de chaîne de requête, accédez à la collection Request.QueryString comme vous le feriez accéder à un cookie.
- La page page.aspx pourrait traiter la chaîne de requête « user » en accédant Request.QueryString["user"] (l'exemple précédent)
- Exemple

```
Sample of Visual Basic Code
Label1.Text = String.Format("User: {0}, Lang: {1}", _
Server.HtmlEncode(Request.QueryString("user")), _
Server.HtmlEncode(Request.QueryString("lang")))
```

```
Sample of C# Code
Label1.Text = string.Format("User: {0}, Lang: {1}",
Server.HtmlEncode(Request.QueryString["user"]),
Server.HtmlEncode(Request.QueryString["lang"]));
```

# Utilisation de la gestion d'état côté serveur

- Dans certains scénarios, le client n'est pas le bon choix pour stocker l'état.
  - Un état qui doit être sécurisé et même crypté et il ne doit pas être transmis sur un réseau.
  - Avoir un état qui n'est pas spécifique à un client mais globale à tous les utilisateurs de votre application.
  - **Solution : utiliser le serveur pour gérer ces états.**

- ASP.NET fournit **deux manières pour stocker l'état sur le serveur** et donc pour partager les informations entre les pages Web sans envoyer les données au client.

# Etat d'application (Application State)

- **État de l'application** dans ASP.NET est un mécanisme de stockage global pour les données d'état qui doit être **accessible à toutes les pages d'une application web, indépendamment de l'identité de l'utilisateur qui demande la page**.

- **Les informations sont perdues si l'application est redémarrée**.

- Vous stockez état de l'application dans une instance de la classe **HttpApplicationState** qui est fourni par la propriété **Page.Application**.
  - Cette classe représente un dictionnaire clé-valeur, où chaque valeur est stockée et elle est accessible par sa clé (ou le nom).
  - Vous pouvez ajouter et lire à partir de l'état d'applications de n'importe quelle page sur le serveur.
  - Cet état reste sur le serveur et n'est pas envoyé au client.

- Toutes les pages peuvent accéder aux données à partir d'un seul emplacement en mémoire, plutôt que de garder des copies séparées des données ou de lire à chaque fois qu'une page est demandée.

# Etat de session (Session State)

- **L'état de session est un état spécifique à un utilisateur**, il est stocké sur le serveur. Il est disponible uniquement aux pages accessibles par un seul utilisateur au cours d'une visite de votre site.
  - Par exemple, si un utilisateur fournit un nom, votre application pourrait le conserver pour le reste de la visite de l'utilisateur en le stockant dans l'état de session.
  - Par exemple, si un utilisateur passe par un processus en plusieurs étapes pour inscrire à votre site, vous pouvez stocker temporairement ces données entre les pages jusqu'à ce que l'utilisateur ait terminé le processus.
- Chaque utilisateur sur votre site a alors son propre isolement état de session fonctionnant dans le processus de l'application sur le serveur.
- Cet état est disponible à différentes pages que l'utilisateur effectue des demandes subséquentes au serveur.
- L'état de session est toutefois perdu si l'utilisateur se termine sa session (ou délai).
- Par défaut, les applications ASP.NET stocke l'état de session en mémoire sur le serveur.

# Lecture et écriture des données de session Etat

- Vous stockez l'état de session spécifique à l'utilisateur dans l'objet de Session.
  - Il s'agit d'une instance de la classe HttpSessionState et représente une collection de dictionnaire valeurs-clés.
- Exemple :

```
Sample of C# Code
// Check if Session object exists, and display it if it does
if (Session["lastVisit"] != null)
{
Label1.Text = ((DateTime)Session["lastVisit"]).ToString();
}
else
{
Label1.Text = "Session does not have last visit information.";
}
// Define the Session object for the next page view
Session["lastVisit"] = DateTime.Now;
```

# Disabling Session State

- Si vous n'utilisez pas l'état de session, vous pouvez améliorer les performances en le désactivant pour l'application toute entière.
- Définir la propriété sessionState mode à Off dans le fichier web.config.

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

- Vous pouvez également désactiver l'état de session pour une seule page d'une application en définissant la directive de page EnableSessionState à false.

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

- Vous pouvez également définir la directive de page EnableSessionState à ReadOnly pour fournir accès en lecture seule aux variables de session pour la page.

# Configuration l'état de session sans cookie (Cookieless)

- Par défaut, l'état de session utilise des cookies pour identifier les sessions des utilisateurs.
  - C'est le meilleur choix pour la grande majorité des applications.
  - Tous les navigateurs modernes prennent en charge les cookies. Cependant, les utilisateurs peuvent les désactiver.
  - Par conséquent, ASP.NET vous permet d'activer l'état de session sans cookie.
- Sans les cookies, ASP.NET suit les sessions en utilisant l'URL, en intégrant l'ID de session dans l'URL après le nom de l'application et avant tout fichier ou identificateur de répertoire virtuel.
  - Par exemple, l'URL suivante a été modifiée par ASP.NET pour inclure l'ID de la session unique lit3py55t21z5v55vlm25s55.
    - http://www.example.com/s(lit3py55t21z5v55vlm25s55)/orderform.aspx
  - Pour activer les sessions sans cookie dans le fichier web.config, il suffit de définir l'attribut cookieless de l'élément sessionState à true.

```
<configuration>
<system.web>
<sessionState cookieless="true"
regenerateExpiredSessionId="true" />
</system.web>
</configuration>
```

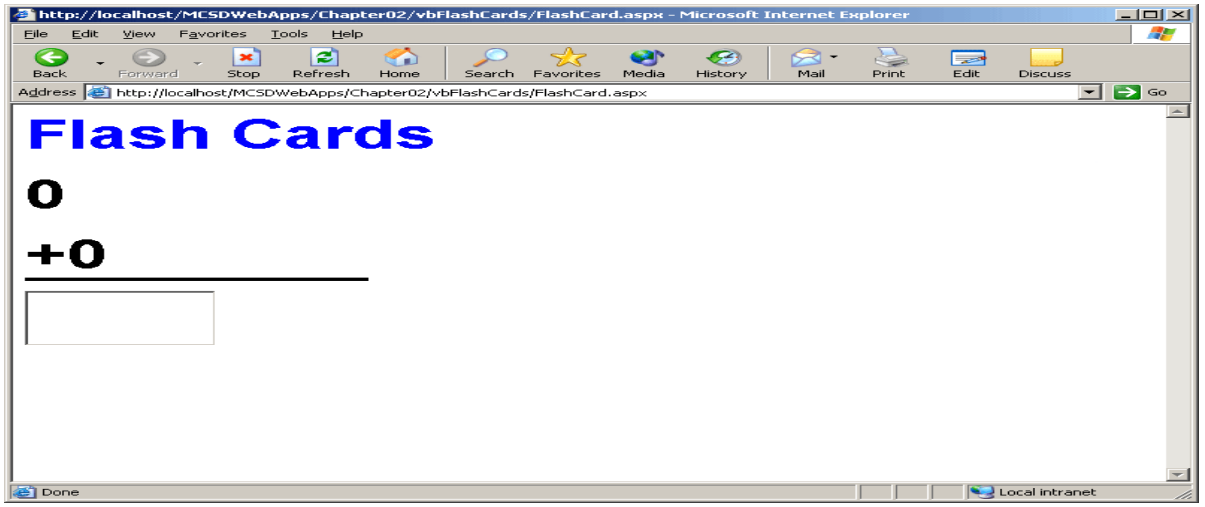
# Répondre aux événements de session

- Souvent, vous voudrez exécuter le code lorsque l'utilisateur ouvre une session ou lorsque la session est terminée.
  - Par exemple, vous voudrez peut-être initialiser les variables clés lorsqu'une session démarre, ou vous voudrez peut-être faire un peu de journalisation spécifique à l'utilisateur.
  - Vous pouvez trapper les événements de session en utilisant le fichier Global.asax.

- Il y a deux événements spéciaux fournies par ASP.NET pour répondre aux activités de la session:
  - session_start : Déclenché lorsqu'un nouvel utilisateur demande une page sur votre site et ainsi commence une nouvelle session. C'est un bon endroit pour initialiser les variables de session.
  - Session_End : Déclenché lorsque la session est abandonnée ou expirée. Cet événement peut être utilisé pour tronçonner l'information ou libérer les ressources par session.

# Architecture Client serveur:
# TD6 - Création d'une application Web simple

- FlashCards est une application Web simple, constitué d'un seul formulaire qui affiche des problèmes mathématiques et évalue les résultats saisis (voir figure).




- Exercice 1: Créer l'interface d'utilisateur

- **Exercice 2: Créer une classe nommée FlashCardClass**
  - Cette classe représente la logique de l'application. Elle permet de générer un problème mathématique d'une manière aléatoire. Cette classe doit contenir au moins les méthodes suivantes :
    - Un constructeur.
    - Une méthode qui permet de générer un problème mathématique.
    - Une méthode qui permet de calculer la réponse du problème mathématique généré.
    - Les méthodes get et set si nécessaire
- **Exercice 3: stocker un objet de la classe FlashCardClass dans l'état de Session**
  - Dans le code de l'événement Page_Load, ajouter le code pour initialiser une variable d'état de session contenant l'objet FlashCardClass que votre formulaire Web utilisera, si c'est la première demande de la page. Sinon, l'objet sera restauré de la session.
- **Exercice 4: Use the FlashCardClass Object from Web Form Events**
  - Dans cet exercice, vous allez écrire le code de la procédure d'événement txtReponse_TextChanged qui permet de vérifier si la réponse entrée est correcte ou non.
  - Si la réponse saisie est correcte alors le message suivant sera affiché : « bravo, votre réponse est correcte ». Ainsi, un nouveau problème mathématique est généré et qui sera affiché et TextBox sera vidé.
  - Sinon, le message suivant sera affiché : « réponse incorrecte, veuillez essayer de nouveau ».