

# Tema 8. ASPNET II

Herramientas Avanzadas para el Desarrollo de Aplicaciones

---

# Indice

1. Controles de lista sencillos
2. Controles de Navegación
3. Controles de Validación
4. Objetos Session y Application
5. Eventos de Aplicación: Global.asax
6. Cookies
7. Envío de email

1

# Controles de lista sencillos

---

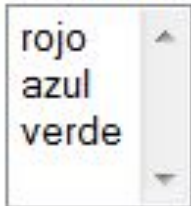
# Controles de Lista Sencillos

## DROPDOWNLIST



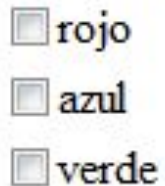
rojo ▼

## LISTBOX



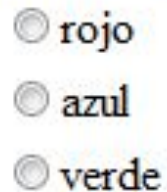
rojo  
azul  
verde

## CHECKBOXLIST



☐ rojo  
☐ azul  
☐ verde

## RADIOBUTTON LIST



☐ rojo  
☐ azul  
☐ verde

• Lista desplegable

• Lista desplegada

• Lista de botones de verificación

• Lista de botones de radio

# Controles de Lista Sencillos (listBox, dropDownList, radioButtonList, checkBoxList)

- Lista desplegable (dropDownList)



- ElementoLista1 (ListItem) **FILA 0**
- ElementoLista2 *Seleccionado* (ListItem) **FILA 1**
- ElementoLista3 (ListItem) **FILA 2**

# Añadir elementos de forma declarativa

```
<asp:DropDownList ID="DropDownList1" runat="server"
    onselectedindexchanged="DropDownList1_SelectedIndexChanged"
    Width="90px" AutoPostBack="False">
    <asp:ListItem Value="rojo1">rojo</asp:ListItem>
    <asp:ListItem>azul</asp:ListItem>
    <asp:ListItem>verde</asp:ListItem>
</asp:DropDownList>
```

# RepeatDirection

- RadioButtonList: lista de botones de radio

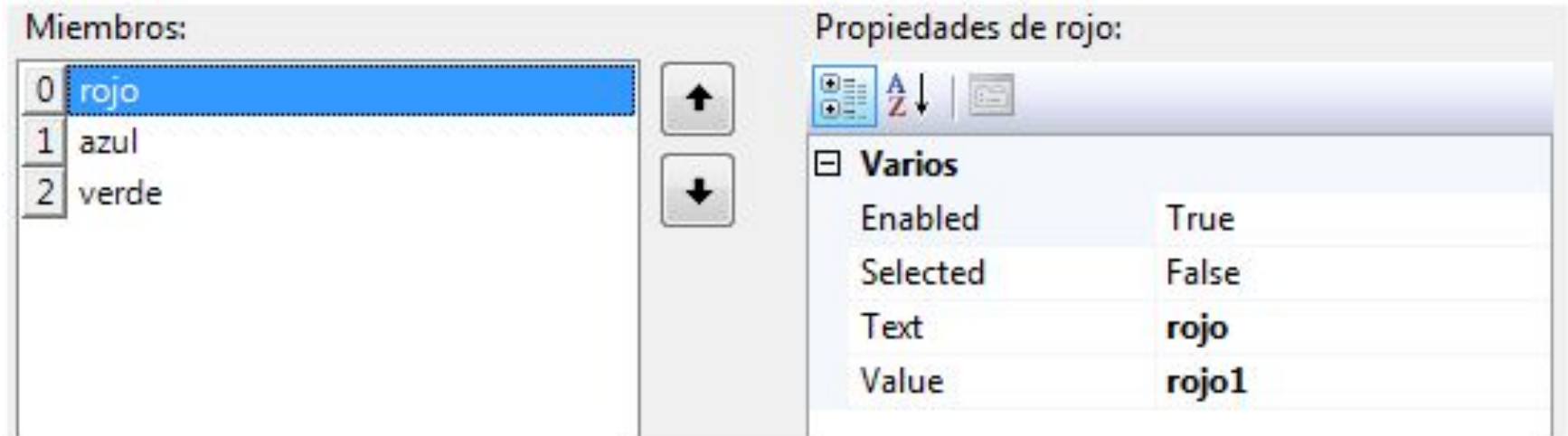
```
<asp:RadioButtonList ID="RadioButtonList1" runat="server"
    onselectedindexchanged="RadioButtonList1_SelectedIndexChanged">
    <asp:ListItem>rojo</asp:ListItem>
    <asp:ListItem>azul</asp:ListItem>
    <asp:ListItem>verde</asp:ListItem>
</asp:RadioButtonList>
```

- CheckBoxList: lista de opciones múltiple

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server"
    onselectedindexchanged="CheckBoxList1_SelectedIndexChanged"
    RepeatDirection="Horizontal">
    <asp:ListItem>rojo</asp:ListItem>
    <asp:ListItem>azul</asp:ListItem>
    <asp:ListItem>verde</asp:ListItem>
</asp:CheckBoxList>
```

# Propiedad Items: colección de ListItem

## Objeto ListItem: propiedades



- Text:
  - contenido visualizado
- Value:
  - valor oculto del código HTML
- Selected:
  - true o false (seleccionado o no)



# Propiedades de los Controles de Lista Sencillos

- Propiedad SelectedIndex
  - Indica la fila seleccionada como un índice que empieza en cero
- Propiedad SelectedItem
  - Permite que el código recupere un objeto **ListItem** que representa el elemento seleccionado

```
Label1.Text = DropDownList1.SelectedIndex.ToString();
```

```
Label1.Text = DropDownList1.SelectedItem.Text.ToString();
```

```
Label1.Text = DropDownList1.SelectedItem.Value.ToString();
```

```
Label1.Text=DropDownList1.SelectedValue;
```

# Controles de lista con selección múltiple

- **ListBox:**
  - Pueden seleccionarse varios elementos: propiedad **SelectionMode=Multiple**
- **CheckBoxList**
  - Siempre pueden seleccionarse varios elementos
- **Hacer:** Para encontrar todos los elementos seleccionados necesitamos
  - recorrer la colección Items del control lista
  - comprobar la propiedad ListItem.Selected de cada elemento

```
Foreach ListItem i in DropDownList1.Items
    { if (i.Selected==true)
      .... }
```

# Añadir elementos dinámicamente a una lista (código)

- Método *Add* del objeto *Items* del control


## EJEMPLO:

```
DropDownList1.Items.Add("rojo");
```

```
DropDownList1.Items.Add(new ListItem("rojo", "Red"));
```



text



value

2

## Controles de navegación: Menu

---

# Control menu

- Se puede utilizar para crear un menú que podemos colocar en la **página maestra**.
- Permite añadir un **menú principal con submenús** y también nos permite definir menús dinámicos.
- Los **elementos del Menu** pueden añadirse directamente en el control o enlazarlos con una fuente de datos.
- En las propiedades podemos especificar la **apariciencia, orientación y contenido del menú**.



# Static Display / Dynamic Display

- El control tiene dos modos de “**Display**”:
  - **Estático (static)**: El control Menu está **expandido completamente todo el tiempo**. Toda la estructura es visible y el usuario puede hacer click en cualquier parte.
  - **Dinámico (dynamic)**: En este caso solo son estáticas las porciones especificadas, mientras que **los elementos hijos se muestran cuando el usuario mantiene el puntero del ratón sobre el nodo padre**.

# Propiedades

- El Control **Menu** es una colección de **MenuItems**
- Propiedades del control → Colección **Items** (colección de objetos **MenuItem**)
- **Añadir objetos individuales MenuItem** (de forma declarativa o programática) .

# Ejemplo

```
<asp:menu id="NavigationMenu" orientation="Vertical" runat="server">
  <items>
    <asp:menuitem navigateurl="Home.aspx"
      text="Home"
      tooltip="Home">
    <asp:menuitem navigateurl="Music.aspx"
      text="Music"
      tooltip="Music">
    <asp:menuitem navigateurl="Classical.aspx"
      text="Classical"
      tooltip="Classical"/>
    <asp:menuitem navigateurl="Rock.aspx"
      text="Rock"
      tooltip="Rock"/>
    <asp:menuitem navigateurl="Jazz.aspx"
      text="Jazz"
      tooltip="Jazz"/>
  </asp:menuitem>
</asp:menuitem>
</items>
</asp:menu>
```



# Otra forma, navegar desde el código C#

```
<asp:Menu ID="Menu1" runat="server" Orientation="Vertical">
  <Items>
    <asp:MenuItem Text="File" Value="File">
      <asp:MenuItem Text="New" Value="New"></asp:MenuItem>
      <asp:MenuItem Text="Open" Value="Open"></asp:MenuItem>
    </asp:MenuItem>
    <asp:MenuItem Text="Edit" Value="Edit">
      <asp:MenuItem Text="Copy" Value="Copy"></asp:MenuItem>
      <asp:MenuItem Text="Paste" Value="Paste"></asp:MenuItem>
    </asp:MenuItem>
    <asp:MenuItem Text="View" Value="View">
      <asp:MenuItem Text="Normal" Value="Normal"></asp:MenuItem>
      <asp:MenuItem Text="Preview" Value="Preview"></asp:MenuItem>
    </asp:MenuItem>
  </Items>
</asp:Menu>
```

# Code-Behind C#

```
protected void Menu1_MenuItemClick(object sender,  
    System.Web.UI.WebControls.MenuEventArgs e)  
{  
    switch(e.Item.Value)  
    {  
        case "File":  
            ...  
            return;  
        case "Open":  
            ...  
            return;  
    }}
```

[http://msdn.microsoft.com/en-us/library/16yk5dbf\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/16yk5dbf(v=vs.80).aspx)

[http://www.obout.com/em/doc\\_server.aspx](http://www.obout.com/em/doc_server.aspx)

3

# Controles de Validación

---

# Validación de datos

- **Debemos asegurar que los usuarios introducen datos correctamente**
  - Dirección de email
  - Número de teléfono
- ASP.Net proporciona un conjunto de **controles de validación** predefinidos
- Dos tipos de validación
  - Validación del **lado del cliente**
  - Validación del **lado del servidor**

# Validación de datos

- **Lado del cliente:**

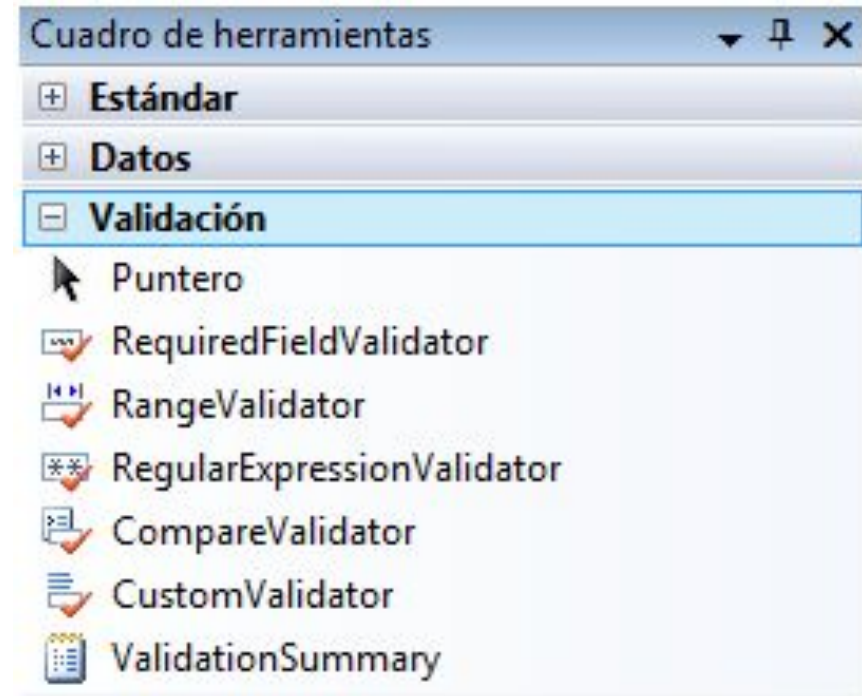
- Utilización de código **JavaScript** que valida los datos introducidos por el usuario, directamente en el navegador

- **Lado del servidor:**

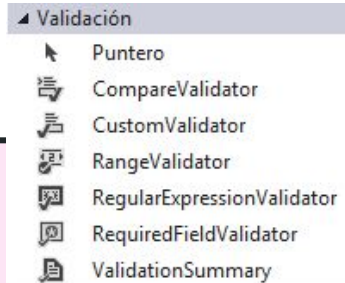
- Utilización de código (**C#**) para validar los datos de formularios una vez han sido enviados al servidor

# Controles de validación

- **ASP.Net detecta si el navegador soporta validación del lado del cliente:**
  - Generan el código **JavaScript** necesario para validar los datos
  - En otro caso, los datos del formulario se validan sólo en el servidor



# Controles de validación



|                            |   |
|----------------------------|---|
| RequiredFieldValidator     | Dato de entrada requerido para el usuario   |
| CompareValidator           | Compara los datos proporcionados por el usuario con un valor constante, o con el valor de otro control (mediante un operador de comparación como menor que, igual que o mayor que) para un tipo de datos específico.  |
| RangeValidator             | Comprueba que una entrada de usuario está entre los límites superior e inferior especificados. Se pueden comprobar los intervalos entre pares de números, caracteres alfabéticos y fechas   |
| RegularExpressionValidator | Comprueba que la entrada del usuario coincide con un modelo definido por una expresión regular. Este tipo de validación permite comprobar secuencias de caracteres predecibles, como los que aparecen en las direcciones de correo electrónico, números de teléfono, códigos postales, etc. |
| CustomValidator            | Comprueba la entrada de usuario utilizando la validación lógica que ha escrito. Este tipo de validación permite comprobar valores derivados en tiempo de ejecución  |

# Controles de Validación

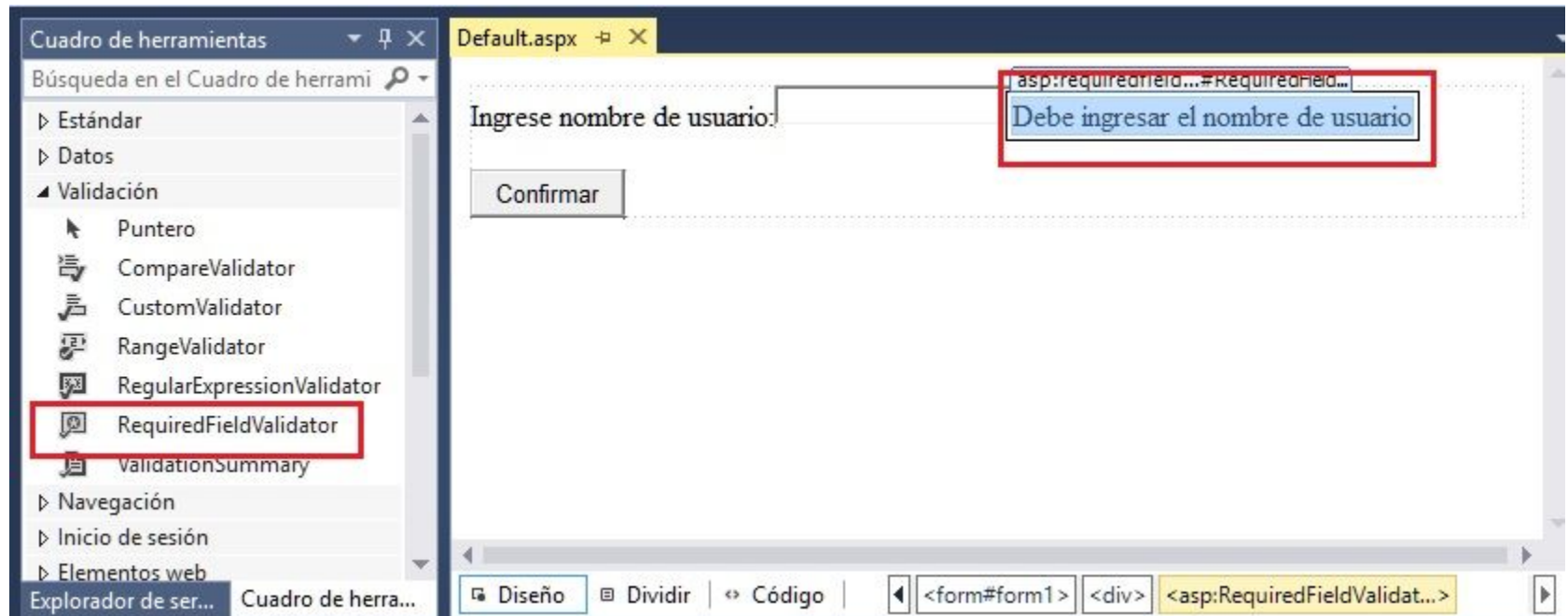
- **Propiedades comunes a todos los controles de validación:**
  - Para mostrar el mensaje de error **ErrorMessage**
  - Para indicar el control a validar **ControlToValidate**
  - Para mostrar texto inicial **Text**
  - **IsValid:** si ha pasado positivamente la validación
- **Propiedad de la página **IsValid****

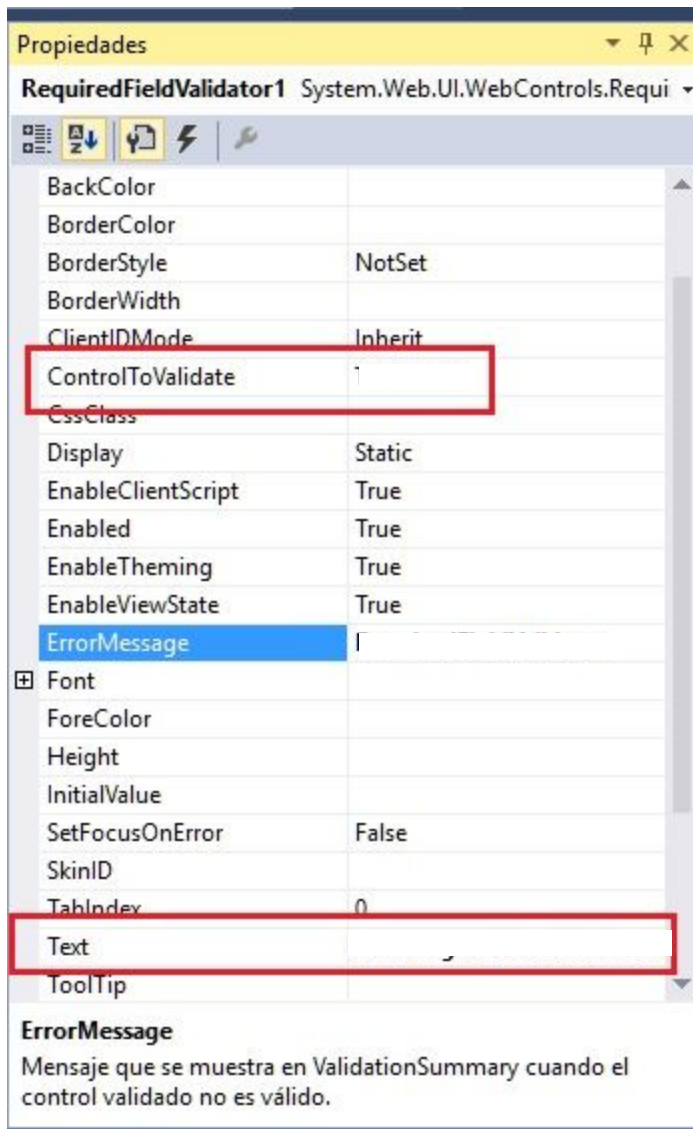
Será true si se pasan las validaciones de la página positivamente



# Entrada requerida

- **Entrada requerida: RequiredFieldValidator:** La validación es OK cuando el control de entrada no contiene una cadena vacía.



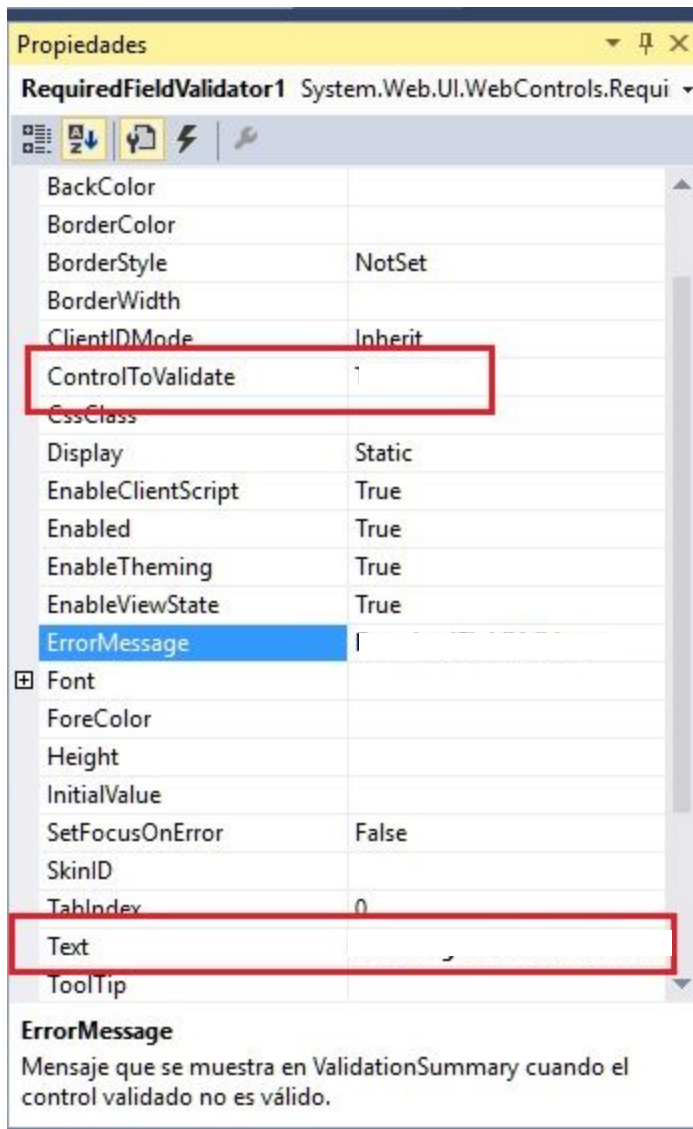


¿Qué ponemos en las propiedades?

ControlToValidate

ErrorMessage

Text



¿Qué ponemos en las propiedades?

**ControlToValidate=**  
TextBox1

**ErrorMessage=** Debe introducir nombre usuario

**Text=** Debe introducir nombre usuario

# Código Button1\_click

Si queremos que al presionar el botón se redireccione a la página Default2.aspx en caso de haber ingresado un nombre de usuario

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
        Response.Redirect("Default2.aspx");
}
```

# Controles de validación

```
<form id="form1" runat="server">
  <div>
    Usuario: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <asp:RequiredFieldValidator ID="UserNameReq" runat="server"
    ControlToValidate="TextBox1" ErrorMessage="Introduce el usuario!!">
    </asp:RequiredFieldValidator>
    Password: <asp:TextBox ID="TextBox2" runat="server"> </asp:TextBox>
    <asp:RequiredFieldValidator ID="PasswordReq" runat="server"
    ControlToValidate="TextBox2" ErrorMessage="Introduce el
    password!!"></asp:RequiredFieldValidator>
    <asp:Button ID="Button1" runat="server" Text="Enviar" />
  </div>
</form>
```

Usuario:  Introduce el usuario!!

Password:  Introduce el password!!

Enviar

# Comprobación de intervalo

- **Comprobación de intervalo: RangeValidator:** La validación es OK cuando el control de entrada contiene un valor dentro de un intervalo numérico, alfabético o temporal especificado.
  - `MaximumValue`
  - `MinimumValue`
  - `Type`
- Definir un control RangeValidator que compruebe si un número está en el intervalo numérico 10–100

Properties

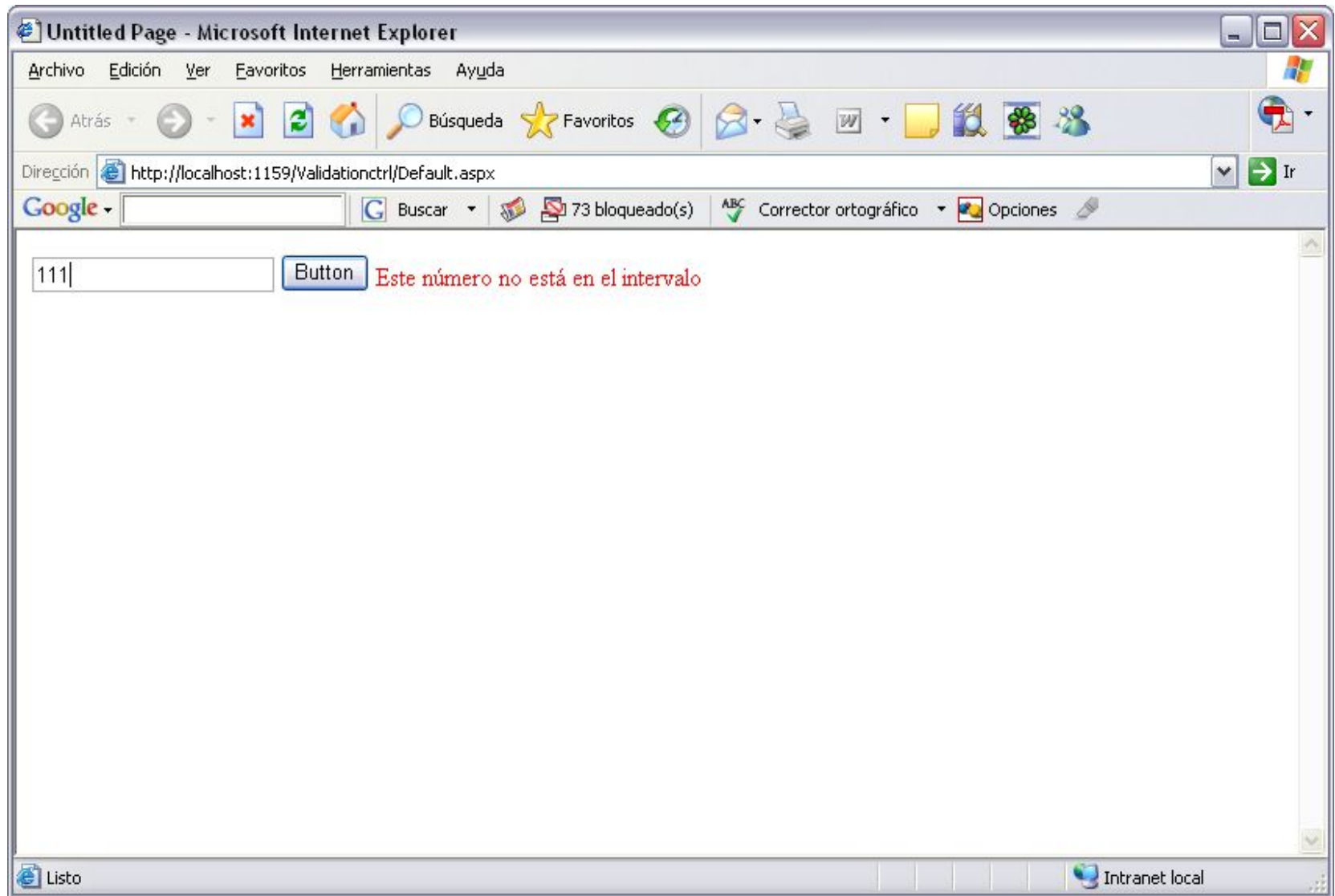
**RangeValidator1** System.Web.UI.WebControls.

(Expressions)

|                        |                                       |
|------------------------|---------------------------------------|
| (ID)                   | <b>RangeValidator1</b>                |
| AccessKey              |                                       |
| BackColor              | <input type="text"/>                  |
| BorderColor            | <input type="text"/>                  |
| BorderStyle            | NotSet                                |
| BorderWidth            |                                       |
| ControlToValidate      | <b>TextBox1</b>                       |
| CssClass               |                                       |
| CultureInvariantValues | False                                 |
| Display                | Static                                |
| EnableClientScript     | True                                  |
| Enabled                | True                                  |
| EnableTheming          | True                                  |
| EnableViewState        | True                                  |
| ErrorMessage           | <b>Este número no está</b>            |
| Font                   |                                       |
| ForeColor              | <input type="color" value="Red"/> Red |
| Height                 |                                       |
| MaximumValue           | <b>100</b>                            |
| MinimumValue           | <b>10</b>                             |
| SetFocusOnError        | <b>True</b>                           |
| SkinID                 |                                       |
| TabIndex               | 0                                     |
| Text                   |                                       |
| ToolTip                |                                       |
| Type                   | <b>Integer</b>                        |

```
<asp:RangeValidator ID="RangeValidator1"
    runat="server"
    ControlToValidate="TextBox1"
    ErrorMessage="Este número no está en
    el intervalo" MaximumValue="100"
    MinimumValue="10" Type="Integer">
</asp:RangeValidator>
```

# Vista ejecución (range validator)





# Comparación

- **Comparación con un valor/control: CompareValidator:** La validación es OK si el control contiene un valor que se corresponde con el valor de otro control especificado.
  - **ControlToCompare / ValueToCompare**
  - **ControlToValidate**
  - **Type**
  - **Operator**

# Ejercicio

- ControlToCompare=
- ValueToCompare=
- ControlToValidate=
- Type=
- Operator=

Default3.aspx\*

Nombre de usuario:

Clave

Repita clave

asp:comparevalidator#CompareValida...  
Las claves ingresadas son distintas

Confirmar

# Solución

- ControlToCompare= TextBox2
- ValueToCompare=
- ControlToValidate= TextBox3
- Type= String
- Operator= Equal

Default3.aspx\*

Nombre de usuario:

Clave

Repita clave

asp:comparevalidator#CompareValida...  
Las claves ingresadas son distintas

Confirmar

# Coincidencia de modelos

- **Coincidencia de modelos: RegularExpressionValidator:** La validación es OK si el valor de un control de entrada se corresponde con una expresión regular especificada.
  - **ValidationExpression**

El control **RegularExpressionValidator** permite validar el valor de un campo de un formulario **con un patrón específico**, por ejemplo un **código postal, un número telefónico, una dirección de mail, una URL** etc.

# Expresiones regulares

- **Dirección de correo electrónico**
  - Comprobar que existe una @, un punto y sólo permite caracteres que no sean espacios.
- **Contraseña**
  - Entre 4 y 10 caracteres y el primer carácter debe ser una letra.
- **Número de cuenta**
  - Secuencia de 4, 4, 2, y 10 dígitos, cada grupo separado por un guión.
- **Campo de longitud limitada**
  - Entre 4 y 10 caracteres incluyendo caracteres especiales (\*, &...)

# Sintaxis Expresiones Regulares

- \* cero o más ocurrencias del carácter o subexpresión anterior.
- + una o más ocurrencias del carácter o subexpresión anterior
- () agrupa una subexpresión que se trata como un único elemento
- | Cualquiera de las dos partes (OR)
- [ ] se corresponde con un carácter en un intervalo de caracteres válidos [a-c]
- {n} exactamente n de los caracteres o subexpresiones anteriores
- . cualquier carácter excepto el salto de línea
- ? el carácter anterior o la subexpresión anterior es opcional
- ^ comienzo de una cadena
- \$ fin de una cadena



- `\s` carácter de espacio en blanco (ej. tab o espacio)
- `\S` cualquier carácter no espacio
- `\d` cualquier carácter numérico
- `\D` cualquier carácter no dígito
- `\w` cualquier carácter alfanumérico (letra, número o carácter de subrayado)

# Solución...

- Correo electrónico
  - `\S+@\S+\.\S+`
- Contraseña
  - `[a-zA-Z]\w{3,9}`
- Número de cuenta
  - `\d{4}-\d{4}-\d{2}-\d{10}`
- Campo de longitud limitada
  - `\S{4,10}`

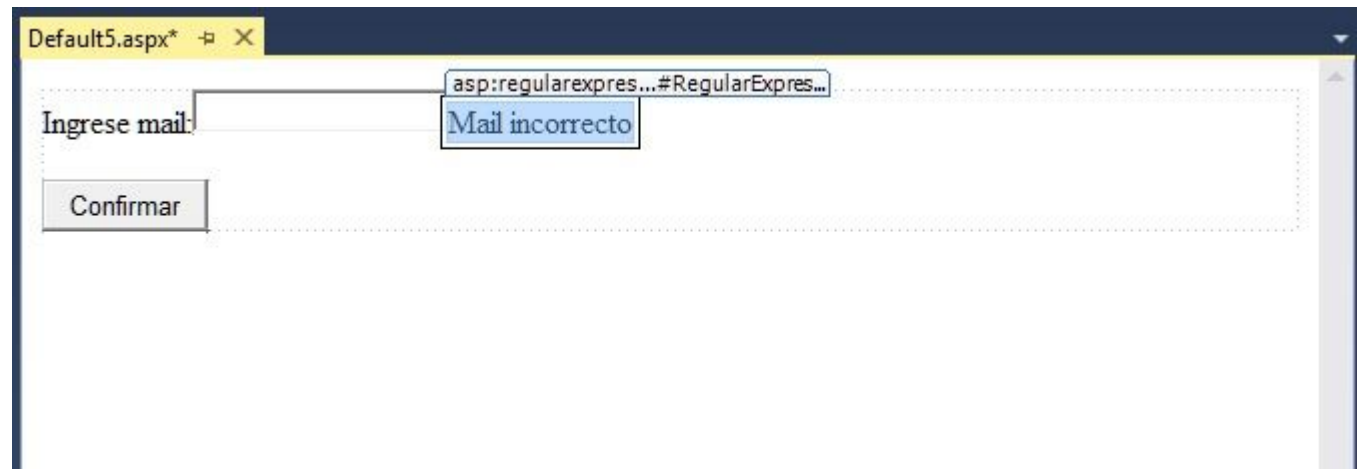


# Propiedades

ControlToValidate=

ErrorMessage=

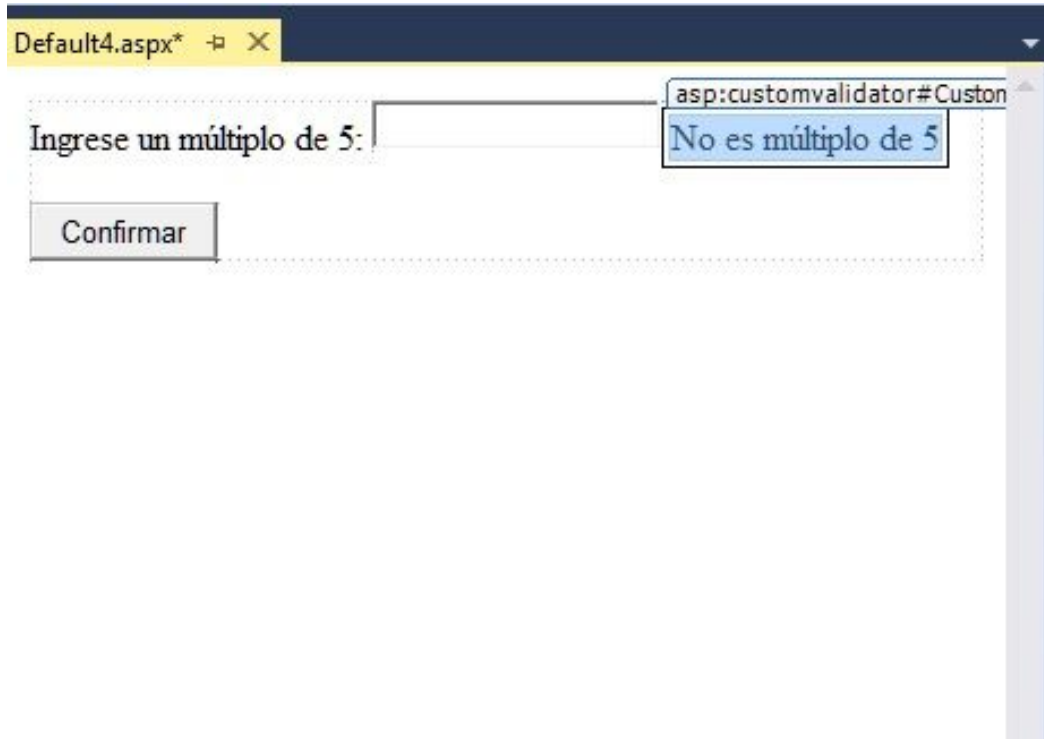
ValidationExpression=



# Controles de Validación

- **CustomValidator**: La validación la realiza una función definida por el usuario.
  - `ClientValidationFunction`
  - `OnServerValidate`
- El control **CustomValidator** permite validar el campo de un formulario con una función de validación propia.
- Debemos asociar nuestro control CustomValidator con un evento propio.

# Ejemplo



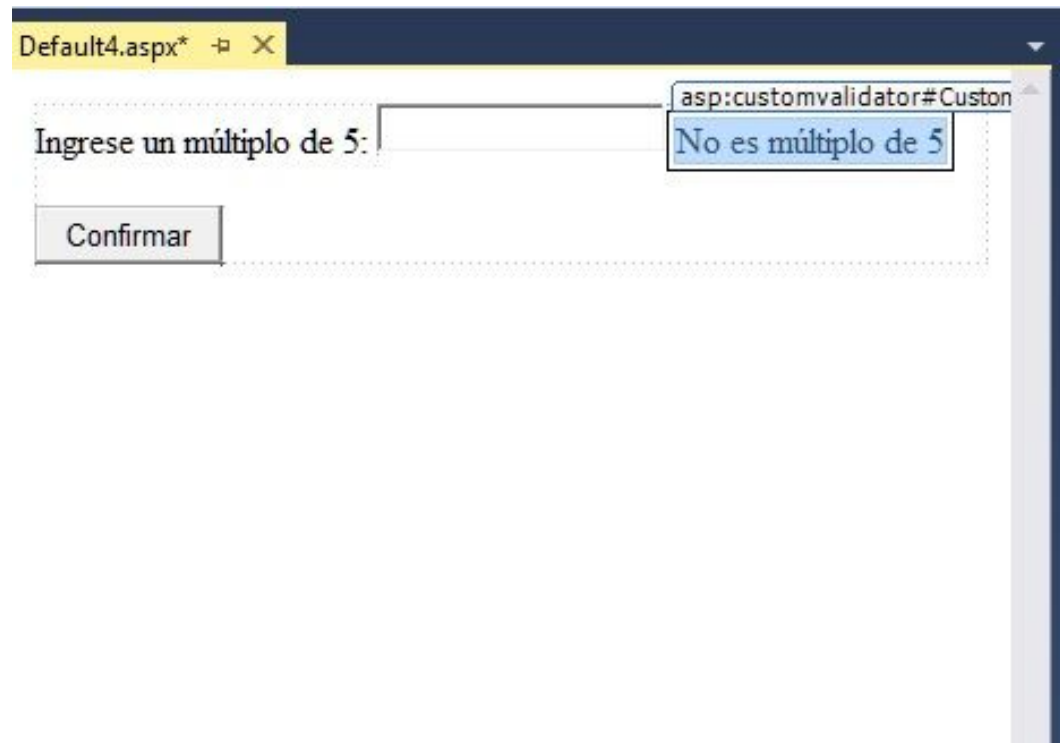
The screenshot shows a web browser window with the title "Default4.aspx\*". The page contains a form with the label "Ingrese un múltiplo de 5:". Next to the label is a text input field. Below the input field is a button labeled "Confirmar". A blue error message box is displayed next to the input field, containing the text "No es múltiplo de 5". The error message box has a title bar that reads "asp:customvalidator#Custom".

página que solicite el ingreso de un número múltiplo de 5, en caso de ingresar un valor incorrecto mostraremos un mensaje de error.

# Propiedades

ControlToValidate=

ErrorMessage=



# C#

```
protected void CustomValidator1_ServerValidate(object source,  
ServerValidateEventArgs args)  
{  
    int valor;  
    valor = int.Parse(TextBox1.Text);  
    if (valor % 5 == 0)  
        args.IsValid = true;  
    else  
        args.IsValid = false;  
}
```

mediante el operador %  
(resto de una división)  
verificamos si es cero

# C#

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        Response.Redirect("Default5.aspx");
    }
}
```

# Controles de Validación

- **ValidationSummary**: Este control muestra un resumen con todos los mensajes de error de cada control de validación.
  - `ShowSummary`

# Ejemplo

La propiedad **Text** de los objetos **RequiredFieldValidator** las inicializamos con un (\*) asterisco y las propiedades **ErrorMessage** con las cadenas: *Debe ingresar el nombre de usuario* y *Debe ingresar la clave* respectivamente

Default6.aspx

body

Usuario:  \*

Clave:  \*

Confirmar

- Mensaje de error 1.
- Mensaje de error 2.

localhost:65278/Default6.aspx

Usuario:  \*

Clave:  \*

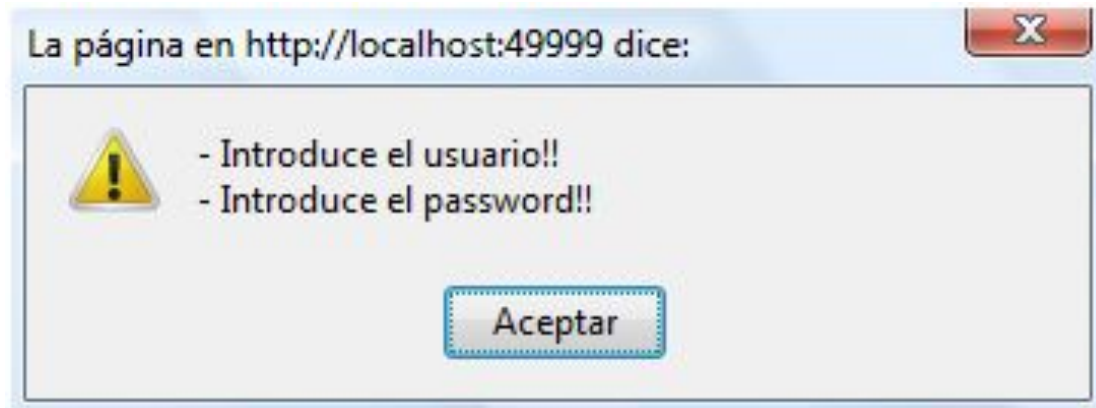
Confirmar

- Debe ingresar el nombre de usuario
- Debe ingresar la clave



# ValidationSummary propiedades

- ShowMessageBox:  
muestra un cuadro  
de diálogo resumen



# El proceso de validación (servidor)

1. El usuario recibe una página y comienza a rellenar los valores de entrada. Al final el usuario pulsa un botón para enviar la página.
2. Cada control Button tiene una propiedad **Causes Validation**.
  - Si esta propiedad es **False**, ASP.NET ignora los controles de validación.
  - Si está a **True** (Valor Predeterminado), ASP.NET valida automáticamente la página cuando el usuario pulsa el botón. Se realiza la validación de cada control de la página.
  - Cada control de validación expone su **propiedad IsValid**, La página también expone una propiedad **IsValid** que resume el estado **IsValid** de todos los controles de validación de la página.

# Añadir a Web.config

```
<appSettings>  
  <add  
    key="ValidationSettings:UnobtrusiveValidationMode"  
    value="None" />  
</appSettings>
```

# 4

Mantenimiento de  
estado:  
Objetos Session y  
Application

---

# Objetos **Session** y **Application**

- Los objetos **Session** están asociados a un usuario particular y sirven como manera de transportar y mantener los datos del usuario en páginas web, como foros o sitios de comercio electrónico.
- Los objetos **Application** son compartidos por todos los usuarios y permiten almacenar información compartida por toda la aplicación web.
- En ASP.NET los objetos **Session** y **Application** están implementados como colecciones o **conjuntos de pares nombre-valor**.

# Qué es una sesión?

- Una **sesión** es el período de tiempo en el que un usuario particular interactúa con una aplicación web.
- Durante una sesión la identidad única de un usuario se mantiene internamente.
- Los datos se almacenan temporalmente en el servidor.
- Una sesión finaliza si hay un *timeout* o si tú finalizas la sesión del visitante en el código.

# Cuál es el uso de una sesión?

- Las sesiones ayudan a preservar los datos entre accesos sucesivos. Esto puede hacerse gracias a los objetos de sesión.
- **Los objetos de Sesión** nos permiten preservar las preferencias del usuario y otra información del usuario al navegar por la aplicación web.
- Ejemplo
- Website de comercio electrónico donde el visitante navega a través de muchas páginas y quiere seguir qué productos ha adquirido.

# Objeto Session

- **Session**: sirve para almacenar datos pertenecientes a un único usuario (en el ámbito de una sesión).

```
//Borra todos los valores de estado de la sesión  
Session.Clear();  
Session.Add("nombre","Homer");  
Response.Write(Session["nombre"]);
```



# En ASP.NET

- Las sesiones son tablas Hash en memoria con un timeout especificado.
- `Session["username"] = "Jose Martínez";`  
`Session["color"] = "Blue";`
- Asignamos los valores a las variables de sesión “username” y “color”, respectivamente. Si necesito saber el “username” o “color” en páginas siguientes puedo usar `Session["username"]`, `Session["color"]`.
- Las sesiones en ASP.NET están identificadas usando enteros 32-bit long conocidos como Session IDs. El motor ASP genera estos session ID's de tal forma que se garantice que son únicos

# Objeto Session

| Session Type             | Qué hace   | Ejemplo  |
|--------------------------|--|--|
| <b>Session.Abandon</b>   | <b>Abandona<br/>(cancela) la sesión<br/>actual</b>                           |  |
| <b>Session.Remove</b>    | <b>Borra un elemento<br/>de la colección de<br/>estado de la<br/>sesión.</b> | <b>Session["username"] =<br/>"Jose Martínez";<br/>(Inicializa una variable de<br/>sesión)<br/>Session.Remove["usernam<br/>"];<br/>(Borra la variable de sesión<br/>"username")</b> |
| <b>Session.RemoveAll</b> | <b>Borra todos los<br/>elementos de<br/>estado de la<br/>sesión.</b>         |  |

| Session Type         | Qué hace   | Ejemplo  |
|----------------------|--|--|
| Session.Timeout      | Establece el the timeout ( <i>en minutos</i> ) para una sesión   | Session.Timeout=30 (Si un usuario NO pide una página en la aplicación ASP.NET en 30 minutos la sesión expira.) |
| Session.SessionID    | Recupera el ID de la sesión (propiedad de sólo lectura de una sesión) para la sesión.  |  |
| Session.IsNewSession | Es para comprobar que la sesión del usuario se creó con la petición actual p.ej. el usuario acaba de entrar al sitio web. La propiedad IsNewSession es cierta en la primera página de la aplicación. |  |

## Ejercicio

- Crear una aplicación web en la cual pidamos introducir un nombre de usuario y un botón enviar.
- Al pinchar en el botón que nos redireccione a un segundo formulario en el cual pondremos “hola “ seguido del login introducido:
  - Metiendo el login en una variable de sesión

# Código

## Default.aspx.cs

```
protected void Button1_Click(object sender, EventArgs e)
{

}
```

## session2.aspx.cs

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

}
```

# Botón con SESSION

Archivo Default.aspx.cs

```
protected void Button1_Click(object sender, EventArgs e)
{
    Session["login"] = TextBox1.Text;
    Response.Redirect("session2.aspx");
}
```

Archivo session2.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    Label3.Text = Session["Login"].ToString();
}
```

# Importante

Al crear la parte privada de la Web utilizaremos variables de sesión para controlar si el usuario ha entrado logueandose o ha entrado poniendo directamente la URL, en cuyo caso la variable de sesión estará vacía y no deberíamos permitir el acceso

# Variables de aplicación

- Y si queremos inicializar variables que estén disponibles en una sesión y sean las mismas para todos los usuarios??
- Esto supone que un cambio en el valor de una **variable de aplicación** se refleja en las sesiones actuales de todos los usuarios.



# Objeto Application

- Por ejemplo:
  - Se puede dar un valor a una variable de aplicación llamada SiteName  
`Application["SiteName"] = "Mi aplicación";`
  - Cualquier página de la aplicación puede leer esa cadena:  
`string appName = (string)Application["SiteName"];`
- Para eliminar cualquier variable del objeto Application:  
`Application.Remove("SiteName");`
- Para eliminar todas las variables:  
`Application.RemoveAll();`

# Variables de aplicación



## Ejercicio

- Crear una aplicación web que cuente el número de visitas que recibe
  - Utilizar variables de aplicación
  - Cuando llegue a 10 visitas el contador se debe reiniciar

# Variables de aplicación

- Primer paso:
  - En la página Default.aspx incluimos una etiqueta  
`<asp:Label ID="LabelCont" runat="server"></asp:Label>`
- Segundo paso:
  - En el archivo Default.aspx.cs (code behind) utilizar una variable Application para controlar el número de visitas

```
protected void Page_Load(object sender, EventArgs e) {  
    if (Application["PageCounter"] != null && (int)Application["PageCounter"] >= 10)  
    { Application.Remove("PageCounter"); }  
    if (Application["PageCounter"] == null)  
    { Application["PageCounter"] = 1; }  
    else  
    { Application["PageCounter"] =  
        (int)Application["PageCounter"] + 1; }  
    LabelCont.Text = Application["PageCounter"].ToString();  
}
```

# Variables de aplicación

- Problema:

- Dos personas cargan simultáneamente la página:

- El contador podría incrementarse sólo 1 unidad

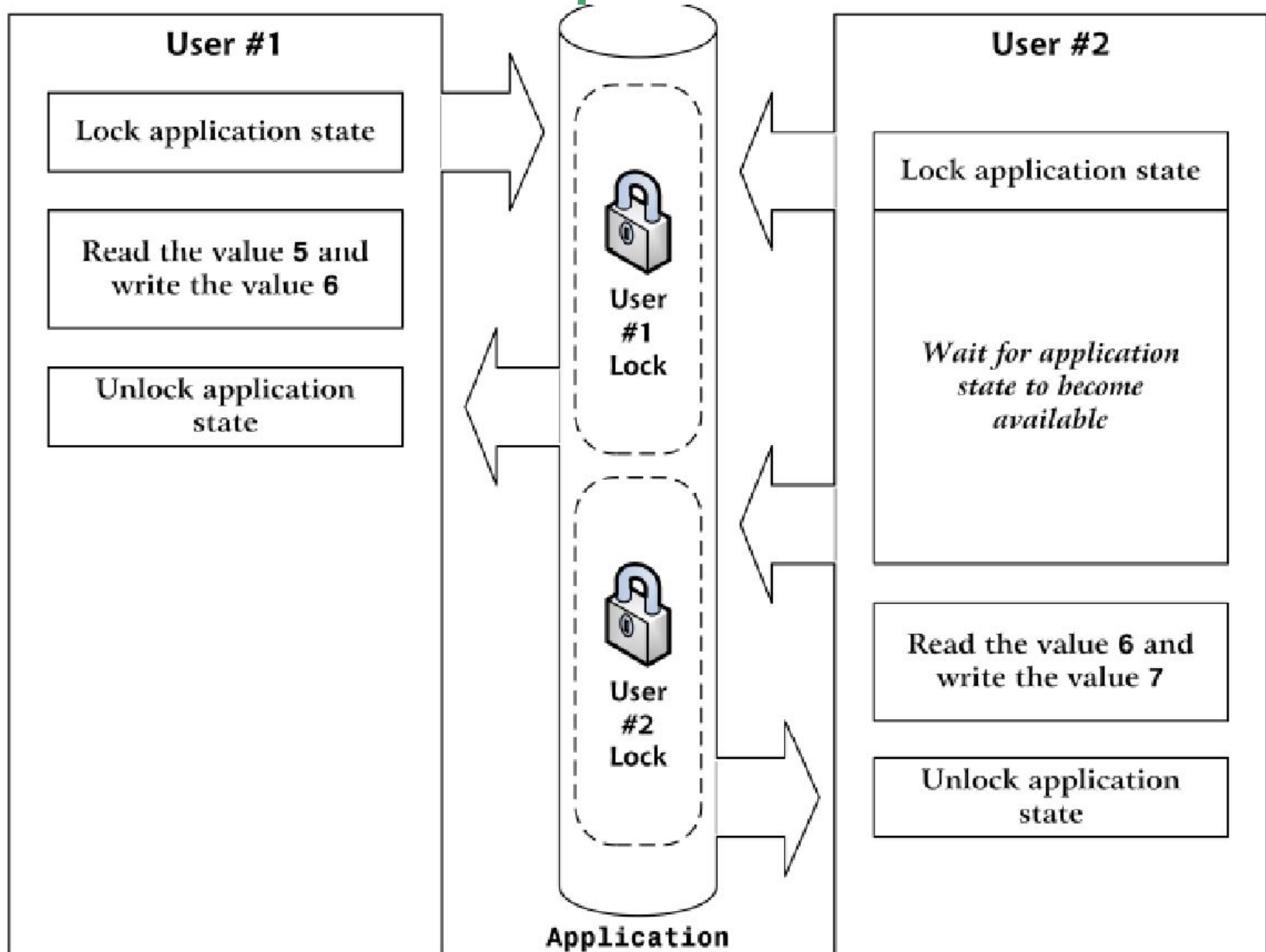
`Application["PageCounter"] = (int)Application["PageCounter"] + 1`

- La expresión de la derecha se evalúa primero
    - El usuario1 lee el valor de PageCounter almacenado en la aplicación
    - El usuario2 lee el valor de PageCounter almacenado en la aplicación
    - Ambos le suman 1 unidad pero el incremento final en lugar de ser 2 unidades es sólo 1

# Actualizar una variable de aplicación

- Solución:
  - En un instante concreto, varias sesiones pueden estar intentando cambiar el valor, aunque sólo una sesión estará autorizada a cambiarlo.
  - ASP.NET tiene exclusión mutua para este tipo de problemas:
    - **Application.Lock()** – Bloquea las variables de aplicación
    - **Application.Unlock()** – Desbloquea las variables de aplicación
  - Una vez que las variables están bloqueadas las sesiones que intentan cambiarlas tienen que esperar.

# Variables de aplicación



# Objeto Application

- **Application:** proporciona una manera sencilla de almacenar en el servidor datos comunes a todos los visitantes de nuestro sitio web.

```
Application.Lock();
```

```
Application.Add("edad",22);
```

```
int valor=(int)Application["edad"];
```

```
valor++;
```

```
Application["edad"]=valor;
```

```
Application.Unlock();
```

```
Response.Write(Application["edad"]);
```

# Problema

- Dónde inicializo una variable de aplicación para que no se reinicie cada vez (ej. Contador de visitas)????



5

Global.asax

---

# El archivo global.asax

- Permite escribir código de aplicación global.
- No contiene etiquetas HTML ni ASP.NET
- Se utiliza para definir variables globales y reaccionar a eventos globales.
- Contiene código de tratamiento de eventos que reacciona a los eventos de aplicación o sesión.

# El archivo global.asax

- Se añade a la aplicación web como un nuevo elemento Clase de aplicación global
  - Global.asax y Global.asax.cs

```
protected void Application_Start(object sender, EventArgs e)
{ Application["SiteName"] = "Mi aplicación"; }
```

```
protected void Session_Start(object sender, EventArgs e)
{
    Session.Timeout = 15;
    Response.Write("Servida el " + DateTime.Now.ToString());
}
```

- Importante:
  - Cualquier cambio en el archivo global.asax reiniciará la aplicación

# Ejemplo, uso de objetos de sesión

- Se quiere modificar el timeout por defecto (20min)
- Se puede hacer en cualquier lugar del código pero lo más recomendable es hacerlo en el archivo Global.asax

## **Archivo Global.asax**

```
protected void Session_Start(object sender, EventArgs e)
{
    Session.Timeout = 15;
}
```

# El archivo global.asax

- Sólo puede haber un archivo global.asax
- Debe residir en el directorio raíz de la aplicación

## Global.asax.cs

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

```
{ Response.Write("<b>");  
    Response.Write("OOps! Ha ocurrido un error! </b>");  
    Response.Write(Server.GetLastError().Message.ToString());  
    Response.Write(Server.GetLastError().ToString());  
    Server.ClearError(); }
```

## Default.aspx.cs

```
int j = 1;  
int x = 0;  
int k=j / x;
```

**OOps! Ha ocurrido un error!** System.Web.HttpUnhandledException: Se produjo una excepción de tipo 'System.Web.HttpUnhandledException'. ---> System.DivideByZeroException:

# CONTADOR DE VISITAS

## Añadir nuevo elemento...

- Clase de aplicación Global → Global.asax

```
void Application_Start(object sender, EventArgs e)
```

```
{
```

```
// Código que se ejecuta al iniciarse la aplicación
```

```
Application.Add("contador", 0);
```

```
}
```

6

Cookies

---

# Cookies

- Para almacenar datos relativos a un usuario se puede utilizar el objeto Session
- Problema:
  - Los datos se borran cuando el usuario cierra la ventana del navegador
- Solución:
  - Para almacenar datos y que éstos se preserven es necesario utilizar las cookies



# Cookies

- Las cookies son extractos de datos que una aplicación ASP.NET puede almacenar en el navegador del cliente para su posterior recuperación
- Las cookies no se pierden cuando se cierra el navegador (a no ser que el usuario las borre)

# Cookies en ASP.NET

- Una cookie se representa por la clase `HttpCookie`
- Las cookies del usuario se leen a través de la propiedad `Cookies` del objeto `Request`
- Las cookies del usuario se modifican a través de la propiedad `Cookies` del objeto `Response`
- Por defecto las cookies expiran cuando se cierra el navegador
  - Se pueden alterar los puntos de expiración (poner una fecha determinada de expiración)

# Cookies en ASP.NET

## Página default.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    HttpCookie userCookie;
    userCookie = Request.Cookies["UserID"];
    if (userCookie == null)
    {
        Label1.Text = "No existe la cookie, creando cookie ahora";
        userCookie = new HttpCookie("UserID", "Ana López");
        userCookie.Expires = DateTime.Now.AddMonths(1);
        Response.Cookies.Add(userCookie);
    }
    else
    {
        Label1.Text = "Bienvenida otra vez, " + userCookie.Value;
    }
}
```

# Cookies en ASP.NET

- La variable `userCookie` se inicializa como una instancia de la clase `HttpCookie` y se le asigna el valor de la cookie llamada `UserID`
- Se comprueba la existencia de la cookie
  - Caso de no existir se muestra un mensaje
  - Se le da el valor “Ana López”
  - Se le asigna una fecha de expiración
- La cookie se transfiere al navegador usando el método `Response.Cookies.Add`
- Si la cookie existe se muestra un mensaje de bienvenida

# Cookies en ASP.NET

- La primera vez que se carga la página se mostrará el mensaje
  - No existe la cookie, creando cookie ahora
- Si se recarga de nuevo la página, la cookie ya existirá y se mostrará el mensaje
  - Bienvenida otra vez, Ana López
- Cuidado!
  - Ten en cuenta que los usuarios pueden rechazar las cookies
  - No puedes dejar que recaigan en ellas aspectos importantes de la aplicación

# Cookies en ASP.NET

- Borrar cookies
  - La única forma es reemplazarla por una cookie con una fecha de expiración que ya ha pasado

```
HttpCookie userCookie= new HttpCookie("UserID");  
userCookie.Expires=DateTime.Now.AddDays(-1);  
Response.Cookies.Add(userCookie);
```

7

Email

—

# Email

- Supongamos que tenemos una tienda online y queremos enviar un email de confirmación de pedido a cada cliente
- En lugar de escribir manualmente cada email ASP.NET permite automatizar este proceso



# Email

- System.Net.Mail
  - **SmtpClient**
  - **MailMessage**
  - **Attachment**
  - **AttachmentCollection**
  - **MailAddress**
  - **MailAddressCollection**

# Email

- **MailMessage**
  - **From**
  - **To**
  - **CC**
  - **Bcc**
  - **Attachments**
  - **Subject**
  - **Body**
  - **IsBodyHTML**

# Email

```
SmtplibClient smtpClient = new SmtplibClient("smtp.gmail.com",587);
MailMessage message = new MailMessage();
try
{
    MailAddress fromAddress = new MailAddress("irene@dlsi.ua.es", "Alias
remidente");
    MailAddress toAddress = new MailAddress("correo@gmail.com", "Alias
destinatario");
    message.Attachments.Add(new Attachment("C:\\imagen1.gif"));
    message.Attachments.Add(new Attachment("C:\\imagen2.jpg"));
    message.From = fromAddress;
    message.To.Add(toAddress);
    message.Subject = "Probando el envío!";
    message.Body = "Este es el cuerpo del mensaje";
    smtpClient.EnableSsl = true;
    smtpClient.Credentials = new System.Net.NetworkCredential("usuario",
"password");
    smtpClient.Send(message);
    Label1.Text = "Mensaje enviado.";
}
catch (Exception ex)
{
    Label1.Text = "No se pudo enviar el mensaje!";
}
```