Departamento de Lenguajes y Sistemas Informáticos

Unit 10. Interface Layer II

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Escuela Politécnica Superior
Universidad de Alicante

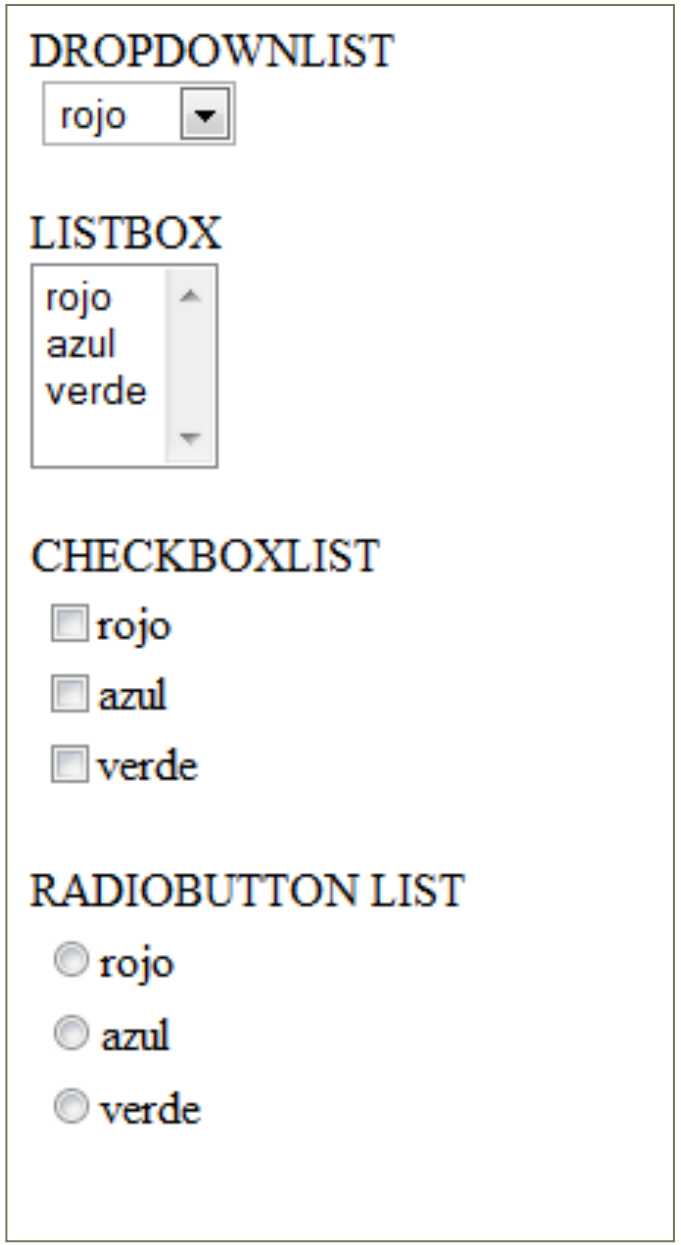
Index

1. Simple list controls
2. Navigation controls
3. Validation controls
4. Session and Application objects
5. Application events: Global.asax
6. Layer distribution in the group project

1

Simple list controls

Simple list controls



Simple list controls (listBox, dropDownList, radioButtonList, checkBoxList)

- DropDownList

- ListItem1 ROW 0
- *Selected* ListItem2 ROW 1
- ListItem3 ROW 2

Adding elements in a declarative way

```
<asp:DropDownList ID="DropDownList1" runat="server"
  onselectedindexchanged="DropDownList1_SelectedIn
  dexChanged" 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

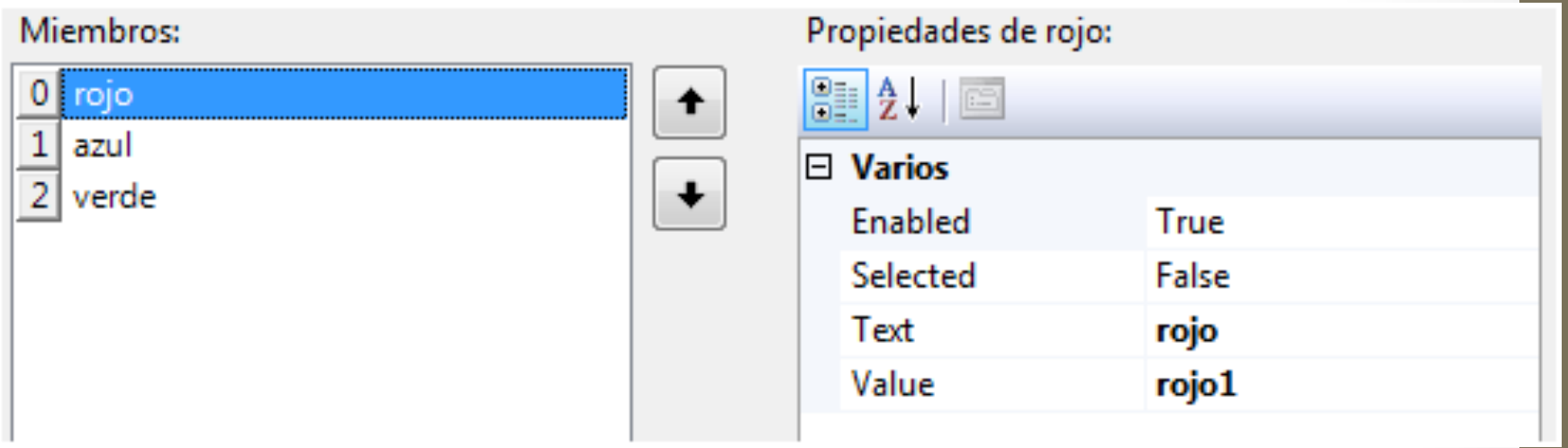
```
<asp:RadioButtonList ID="RadioButtonList1" runat="server“
   onselectedindexchanged="RadioButtonList1_SelectedIndexChanged">
         <asp:ListItem>rojo</asp:ListItem>
         <asp:ListItem>azul</asp:ListItem>
         <asp:ListItem>verde</asp:ListItem>
</asp:RadioButtonList>
```

- CheckBoxList

```
   <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>
```

Items property: ListItems collection
ListItem Object: properties



- Text:
  - Visualized content
- Value:
  - Hidden value of the HTML code
- Selected:
  - true or false

Properties of the simple list controls

- SelectedIndex
  - Indicates the selected row as an index that starts by zero
- SelectedItem
  - Allows that the code gets a ListItem object that represents the selected item

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

List controls with multiple choice

- ListBox:
  - We can select several elements: property **SelectionMode=Multiple**
- CheckBoxList
  - We can always select several elements

- In order to know which are the selected elements we need
  - going over the items collection of the list control
  - Checking the property ListItem.Selected of each item

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

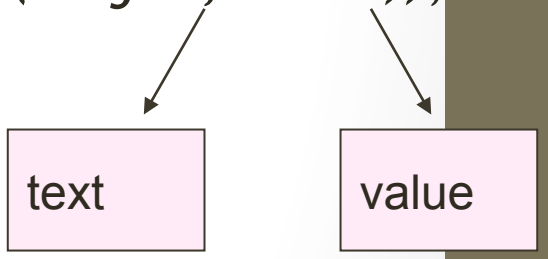
Adding elements dynamically to a list (code)

- Method *Add* of the *Items object*

EJEMPLO:

DropDownList1.Items.Add("rojo");

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



2 Navigation controls: Menu

Menu control

- We can use it to create a menu that we will place in the master page.
- We can add a main menu with submenus and we can also add dynamic menus.
- The Menu items can be added directly to the control or binded with a data source.
- In the properties we can specify the look, orientation and content of the menu.

Static Display and Dynamic Display

- Two "**Display**" modes:
  - **Static**: The control is expanded all the time. All the structure is visible and the user can click in any place.
  - **Dynamic**: In this case, the static parts are specified, but the children elements are shown when the user keeps the mouse on the parent node.

Properties

- StaticDisplayLevels.
  - This property allows to specify the number of static levels we want to show as root of the menu (The minimum is 1).

- MaximumDynamicDisplayLevels
  - This property specifies the number of node levels that appear in a dynamic way after the static level.

Dynamic: amount of time

- We can also specify the amount of time that we want the dynamic part lasts in dissapearing.
- We can specify it in miliseconds with the property **DisappearAfter** of the menu:
  - Menu.DisappearAfter=1000;
- The default value is 500 ms.

Defining the content

- Adding individual MenuItem items (declaratively or programatically) . We can also bind a XML file.
- Properties of the control→ Items property (collection of MenuItem objects)

```
<asp:Menu ID="Menu1" runat="server" StaticDisplayLevels="3">
  <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 "Products":
      ...
      return;
    case "Services":
      ...
      return;
  }}
```

http://msdn.microsoft.com/en-us/library/16yk5dby(v=vs.80).aspx

http://www.obout.com/em/doc_server.aspx

3

Validation controls

Data Validation

- We have to check that the user inputs are correct
  - Email address
  - Telephone number

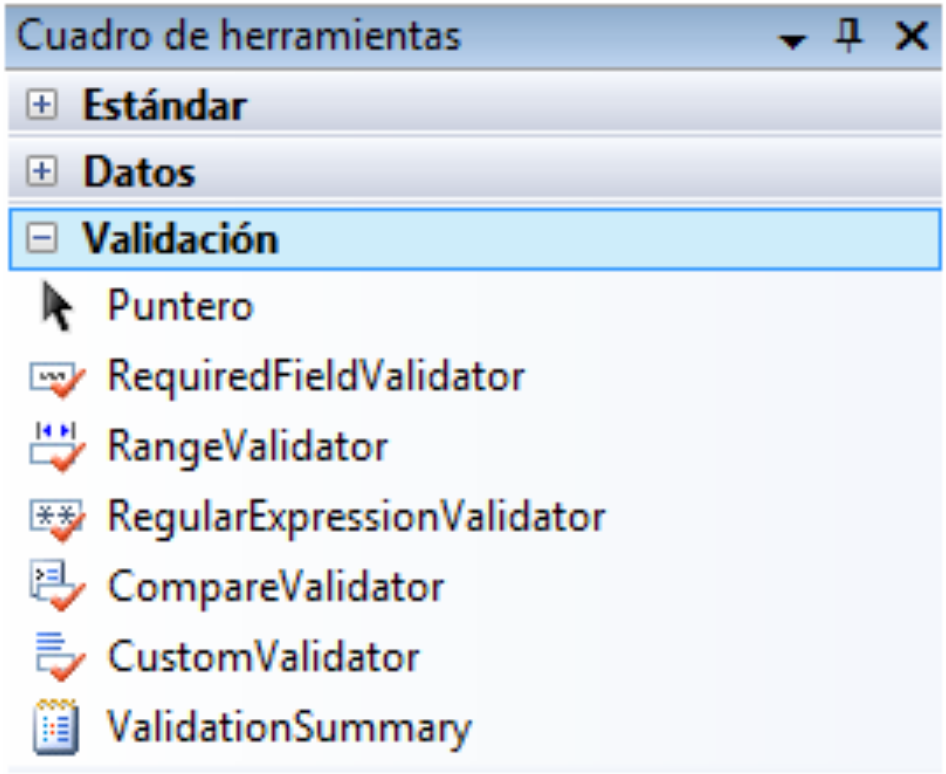
- ASP.Net provides a set of predefined validation controls
- Two types of validation
  - Client side
  - Server side

Data Validation

- Cient side:
  - Using JavaScript code that validates the data introduced by the user, directly on the browserr
- Server side:
  - Using C# code to validate the data of the forms once they are sent to the server

Validation controls

- ASP.Net detects if the browser supports client side validation:
  - They generate the needed JavaScript code
  - In other case, the data of the form are validated on the server



Validation controls

- Over different web controls:
  - TextBox
  - ListBox
  - DropDownList
  - RadioButtonList
  - FileUpload

Validation controls

- Properties
  - To show the **ErrorMessage**
  - To indicate the **ControlToValidate**
- **Required input:** **RequiredFieldValidator:** the validation is OK when the input control does not contain an empty string.
  - InitialValue
- **Model matching:** **RegularExpressionValidator:** the validation is OK if the input control value matches with a specified regular expression.
  - ValidationExpression

Validation controls

```
<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>
```

Validation controls

- Comparing with a value: CompareValidator: The validation is OK if the control contains a value that corresponds with the value of other specified control.
  - ControlToCompare
  - ControlToValidate
  - ValueToCompare
  - Type
  - Operator
- Checking of a range: RangeValidator: The validation is OK when the input control contains a value inside a specified numberic, alphabetic or temporal range. MaximumValue
  - MinimumValue
  - Type

Validation controls

- **CustomValidator:** The validation is done by a user defined function.
  - ClientValidationFunction
  - OnServerValidate
- **Let's suppose that we want to validate that the user name is unique in the application**

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:CustomValidator ID="CustomValidator1“
    ControlToValidate="TextBox1“
    OnServerValidate="ComprobarUsuario" runat="server"
    ErrorMessage="El usuario ya existe!!">
</asp:CustomValidator>
```

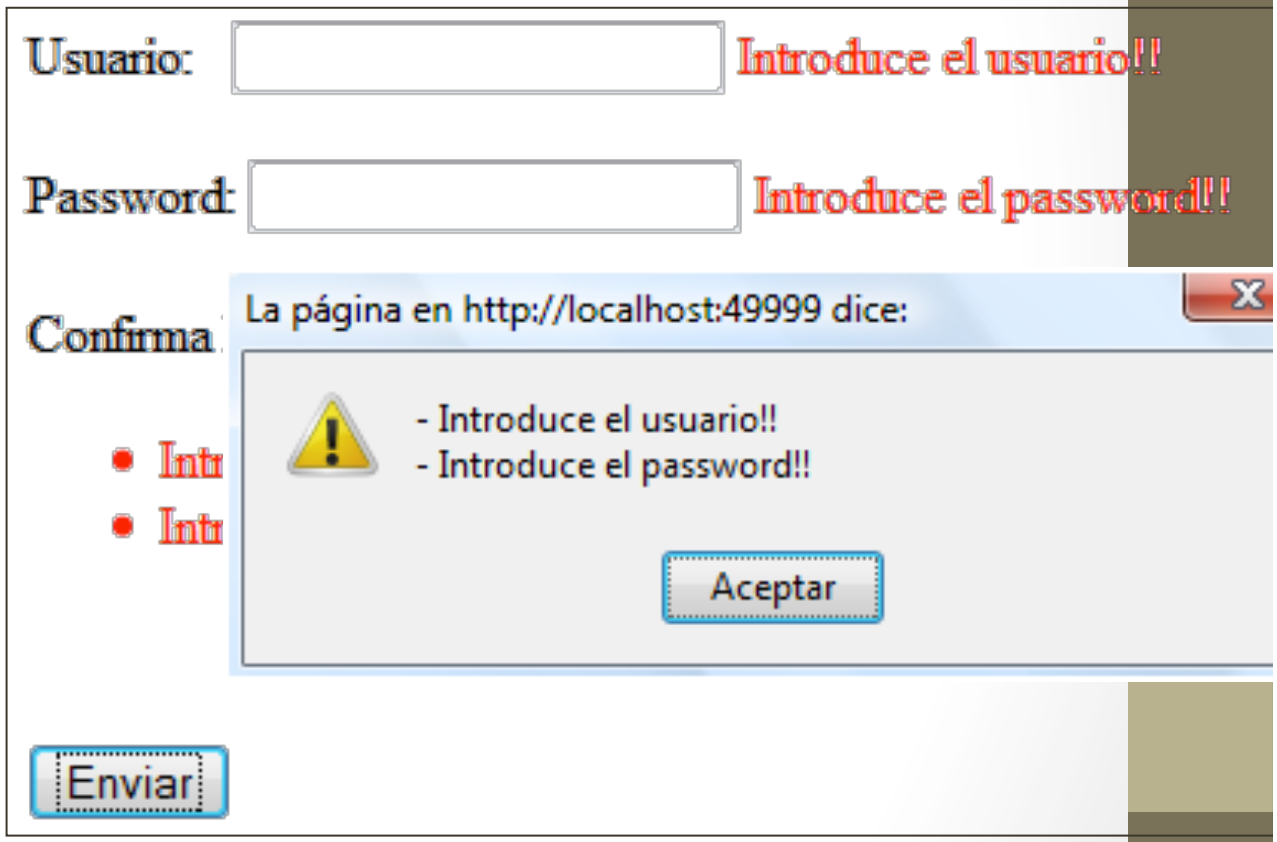
Validation controls

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {   Button1.Text = "Correcto"; }
    else
    {   Button1.Text = "Incorrecto"; }
}

protected void ComprobarUsuario(object sender,
  ServerValidateEventArgs e)
{
    string username = e.Value.ToLower();
    if (username == "sonia" || username == "irene")
    { e.IsValid = false; }
}
```

Validation controls

- **ValidationSummary:** This control shows a summary with all the error messages of each validation control.
  - ShowMessageBox
  - ShowSummary



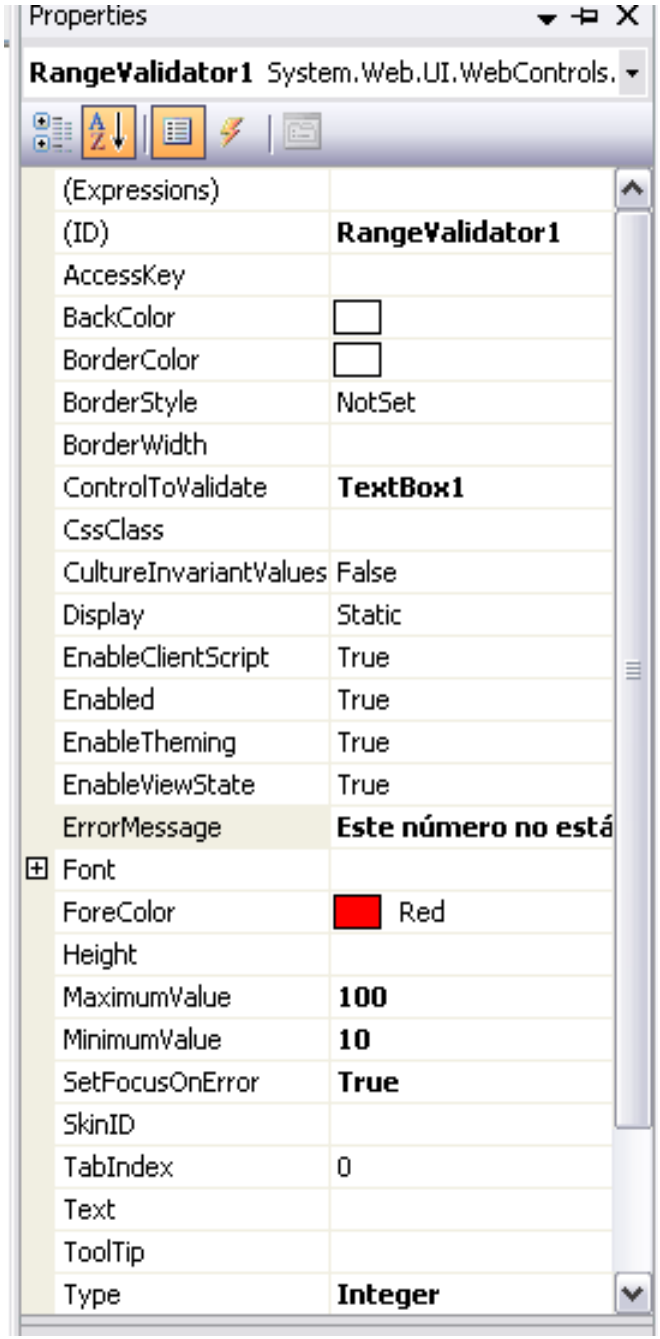
Display Property

- It can have different values :
- **None** — The validation control does not appear in the page.
- **Static** — Each validation control takes a place on the page even if the text of the error message is not visible, what allows defining a fixed location on the page.
- **Dynamic** — The validation controls don't occupy space unless they show an error message, so they can share the same place on the page. However, when we show the error message, the layout of the webpage changes, so sometimes the controls change of location.

- The dynamic layout requires a browser with Dynamic Html support (DHTML).

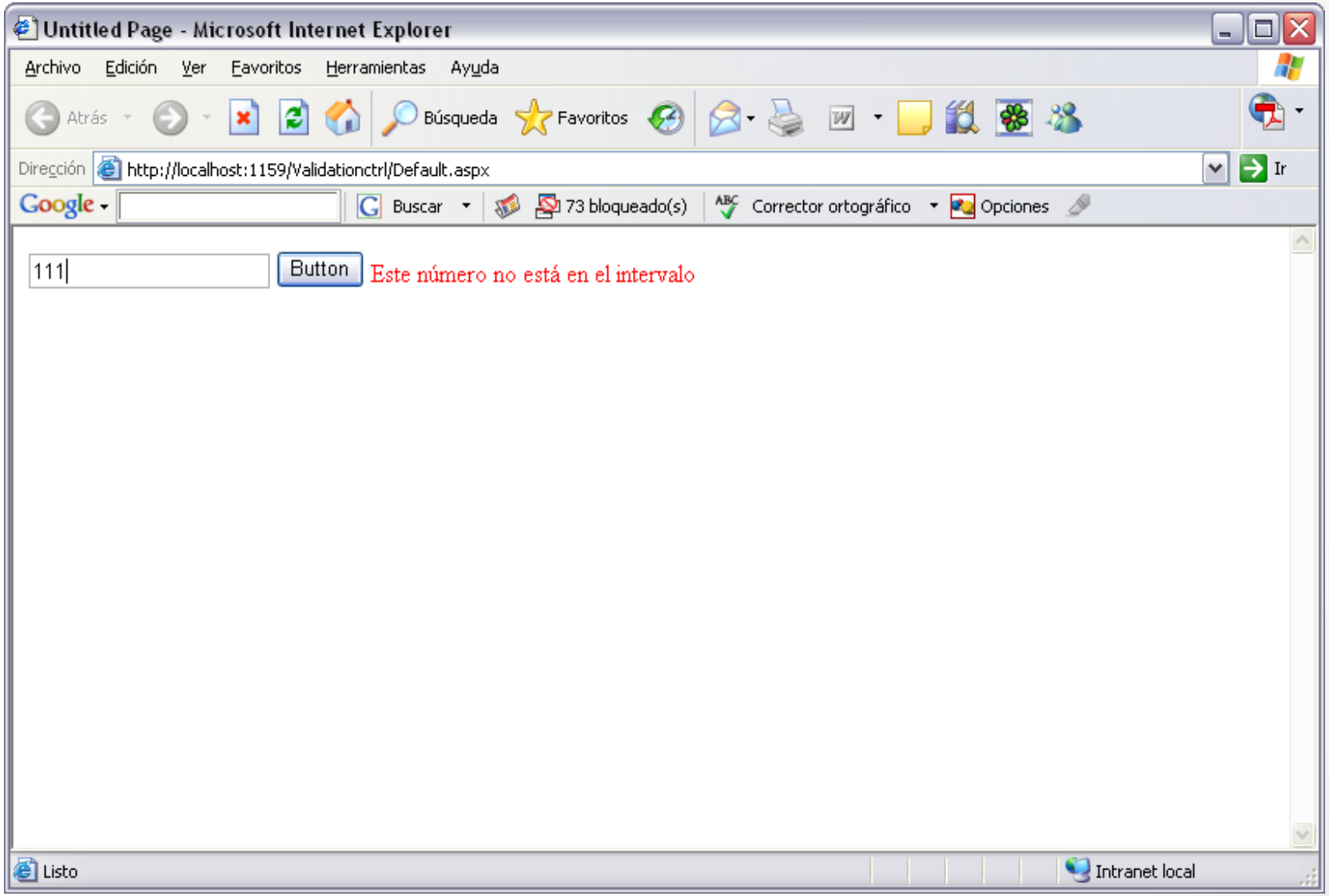
The validation process (server)

1. The user gets a page and start filling in the input values. Finally the user presses a button to submit the page.
2. Each Button control has a *Causes Validation property*.
   - If it is set to False, ASP.NET ignores the validation controls.
   - Iff it is set to True (Default value), ASP.NET automatically validates the page when the user presses a button. The validation of each of the controls is performed.
   - Each validation control shows its **IsValid** property**.** The page also has a IsValid property that summarizes the **IsValid** status of each of the validation controls of the page.




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

Execution time (range validator)



Regular expressions

- **Email address**
  - Check that exists a @, one dot and admit only the non-space characters.
- **Password**
  - Between 4 and 10 characters, and the first one must be a letter.
- **Account number**
  - Sequence of 4, 4, 2, and 10 digits, each group separated by a dash.
- **Limited size field**
  - Between 4 and 10 characters, including special characters (*, &…)

Syntax of Regular Expressions

- * zero or more occurrences of the previous character or subexpression.
- + zero or more occurrences of the previous character or subexpression.
- () groups a subexpresion that is treated as a unique element
- | Each of the two parts(OR)
- [ ] it corresponds with a character on a range of valid characters [a-c]
- {n} exactly n of the previous characters or subexpresions
- . Each character except the line feed
- ? The previous character or subexpression is optional
- ^ beginning of a string
- $ end of a string

. . .

- \s white space character (e.g. tab or space)
- \S any non-space character
- \d any numeric character
- \D any non-digit character
- \w any alphanumberic character (letter, number or underline character)

Solution…

- Email address
  - \S+@\S+\.\S+
- Password
  - [a-zA-Z]\w{3,9}
- Account number
  - \d{4}-\d{4}-\d{2}-\d{10}
- Limited size field
  - \S{4,10}

Validation groups

- The validation controls can be associated in validation groups with the aim that the ones belonging to the same group are validated together.
- These validation groups can be used to enable or disable in a selective way the validation of related controls in a page.

- We have to define a name of the group in the validation controls and the button or other sending controls that cause the validation.

SetFocusOnError

- We can set this property on the validation controls that cause that the first invalid control gets the focus.

- Property of the validation controls.

- More information of these controls

http://msdn.microsoft.com/es-es/library/debza5t0(v=vs.90).aspx

http://www.elguille.info/colabora/NET2005/FernandoLuque_ASPValidar.htm

4

Status maintenance: Session and Application Objects

Session and Application Objects

- Session objects are related to a particular user and are a way of sharing and maintaining user data between web pages, such as forum or e-commerce websites.
- Application objects are shared among all the users, and allow storing shared information in the entire web application.
- In ASP.NET the Session and Application objects are implemented as collections or sets of name-value pairs.

What is a session?

- It is the period of time that a particular user interacts with a web application.
- During a session, the unique identity of a user is internally maintaned.
- The data are temporary stored in the server.

- A session finishes if there is a *timeout* or if we finish the user session in the code.

What is the use of a sesion?

- Sessions help to preserve the data in following accesses. This can be done thanks to the session objects.
- **Session objects help us to preserve the user preferences and other user related information when browsing the web application.**

- Example
- E-commerce Website where the visitor browses may web pages and wants to know the products bought.

Session object

- Session: it is used to store data belonging to a unique user (in a session scope).

```
//Delete all the status values of the session
Session.Clear();
Session.Add("nombre","Homer");
Response.Write(Session["nombre"]);
```

In ASP.NET

- Sessions are Hash tables in memory with a specified timeout.
- Session[“username”] = “Jose Martínez”;
  Session[“color”] = “Blue”;

- We assign the values to the session variables “username” and “color”, respectively. If I need to know the “username” or “color” in the following pages I can use Session[“username”], Session [“color”].

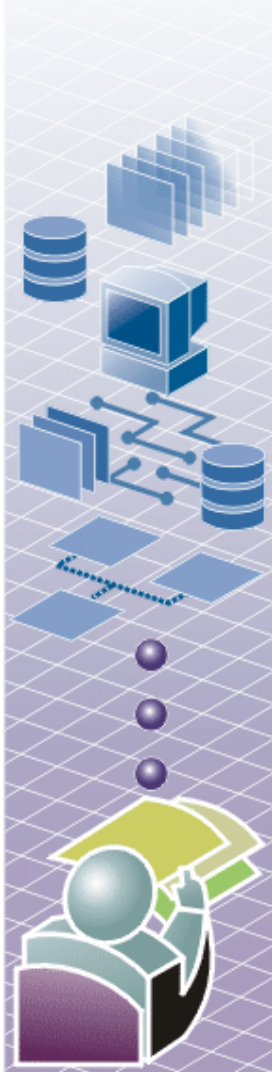
- The sessions in ASP.NET are identified using 32-bit long integers known as Session IDs. The ASP engine generates those session ID’s assuring that are unique.

Session Object

Session Type	Description	Example
Session.Abandon	Cancels the current session	
Session.Remove	Deletes an element of the status collection of the session	Session[“username”] = “Jose Martínez”; (Initiates a session variable) Session.Remove[“username”]; (Deletes the session variable “username”)
Session.RemoveAll	Deletes all the elements of the session	

Session Type	Description	Example
Session.Timeout	Stablishes the timeout (*in minutes*) for a session	Session.Timeout=30 (If a user does NOT request a page in the ASP.NET application in 30 minutes, the session expires.)
Session.SessionID	Recovers the ID of the session (read-only property of a session).	
Session.IsNewSession	It is used to check that the user session was created with the current request, e.g. the user just entered the website. The IsNewSession property is true in the first page of the application.	

Exercise



Exercise

- Create a web application where we ask for a user name and a submit button.
- When clicking the button it will redirect us to a second page where we will write “hola “ followed by the introduced login,:
  - Using session variables

Solution

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();
}
```

Important

When creating the private part of the Web we are going to use session variables to check if the user has entered by doing login or directly typing the URL, in which case the session variable would be empty and we shouldn't allow the access.

Application variables

- And if we want to initialize variables that are available during a session and are the same for all the users??

- This causes that a change in an application variable is reflected in all the current sessions of all the users.

Application Objects

- For instance:
  - We can give value to an application variable called SiteName

```
Application["SiteName"] = "Mi aplicación";
```

  - Every page of the application can read that string:

```
string appName = (string)Application["SiteName"];
```

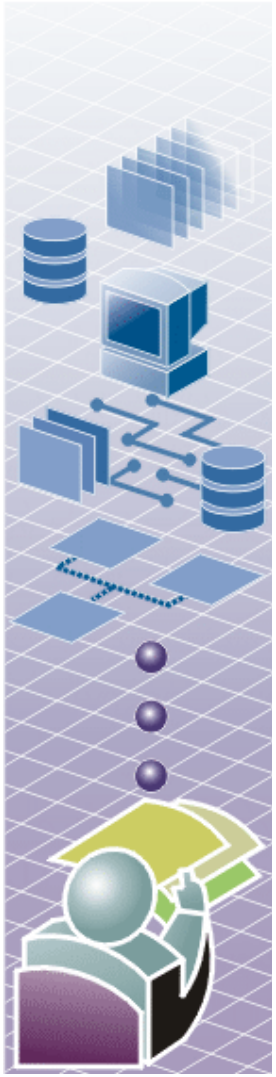
- In order to remove any variable of the Application object:

```
Application.Remove("SiteName");
```

- And for removing all the variables:

```
Application.RemoveAll();
```

Application variables



Exercise

- Create a web application that counts the number of visits
  - Use application variables
  - When we get10 visits the counter should be restarted

Application variables

- First step:
  - In the page Default.aspx we include a label

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

- Second step:
  - In the file Default.aspx.cs (code behind) use an Application variable to control the number of visits

```
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();
}
```

Application variables

- Problem:
  - Two persons load the page at the same time:
    - The counter could be increased only in one unit

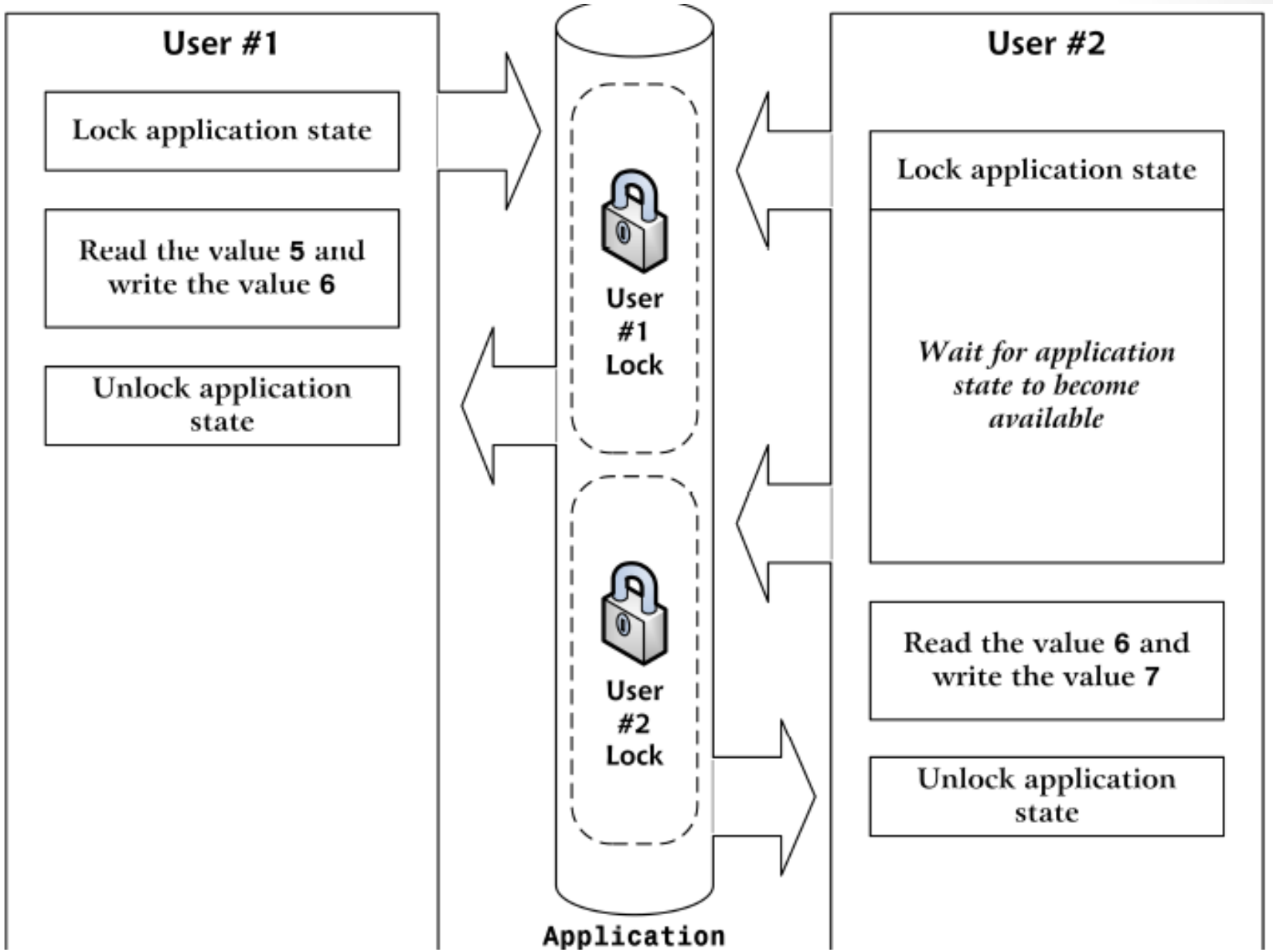
```
Application["PageCounter"] = (int)Application["PageCounter"] + 1
```

    - The right hand expression is evaluated first
    - The user1 reads the value of the PageCounter stored in the application
    - The user2 reads the value of the PageCounter stored in the application
    - Both increase one unit the counter, but the final increasement instead of being 2 units is only 1

Updating an application variable

- Solution:
  - In a concrete moment, several sessions can be trying to update the value, but only one can do it.
  - ASP.NET has mutual exclusion for these type of problems:
    - **Application.Lock()** – Locks the application variables
    - **Application.Unlock()** – Unlocks the application variables
  - Once the variables are lock, the sessions that try to update them will have to wait.

Application variables



Application object

- **Application**: provides a simple way of storing in the server common data to all the visiors of our website.

```
Application.Lock();
Application.Add("edad",22);
int valor=(int)Application["edad"];
valor++;
Application["edad"]=valor;
Application.UnLock();
Response.Write(Application["edad"]);
```

Problem

- Where do I initialize an application variable so it is not instantiated each time (eg. Visitor counter)????

5 Global.asax

The global.asax file

- It allows to write global application code.
- It does not contain HTML or ASP.NET tags
- It is used to define global variables and react on global events.
- It contains code for handling events that react to the application or session events.

The global.asax file

- We can add this file to the Web application as a new element of the type Global application

```
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());
 }
```

- Application_Start is executed the first time the application runs (or just after we restart the server).

- Important:
  - Any change in the global.asax file will cause the application to re-start

Example, use of session objects

- We want to modify the default timeout (20min)
- We can do it any place in the code, but is more recommended doing it in Global.asax

Archivo Global.asax

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

Global.asax

- It can only exist one global.asax file
- It should be in the root directory of the application

Default.aspx.cs

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

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();   }
```

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

VISITOR COUNTER
Adding new element…

- 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 Layer distribution in the project

In the group assignment…

- A solution with 2 projects:
  - Web project(with a reference to the library project)
    - It will contain the interface and the validations (and the connection with the BE)
  - Class library project with the layers BE and DAC
    - BE folder
      - BE Classes of the system entities
    - DAC folders
      - For every BE a DAC class for the data access