

Departamento de Lenguajes y Sistemas Informáticos

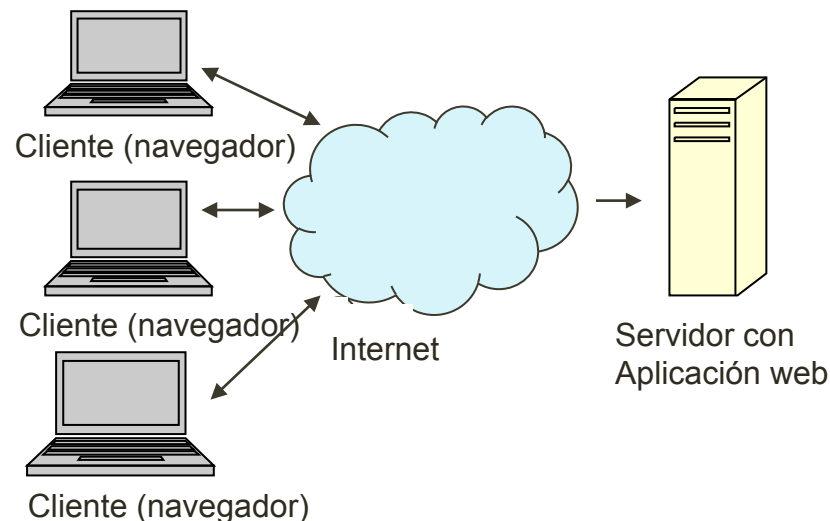
Tema 9. Aplicaciones Web. Capa de Interfaz (I)

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Escuela Politécnica Superior
Universidad de Alicante

Aplicaciones web vs escritorio

- Simple creación de interfaz de usuario.
- Distribución de actualizaciones más fácil y rápido y menos costoso.
- Procesamiento distribuido.
 - Muchísimo más fácil proveer procesamiento del lado del servidor.
 - La web provee protocolos estándar (HTTP, HTML, XML) para facilitar aplicaciones n-capa.



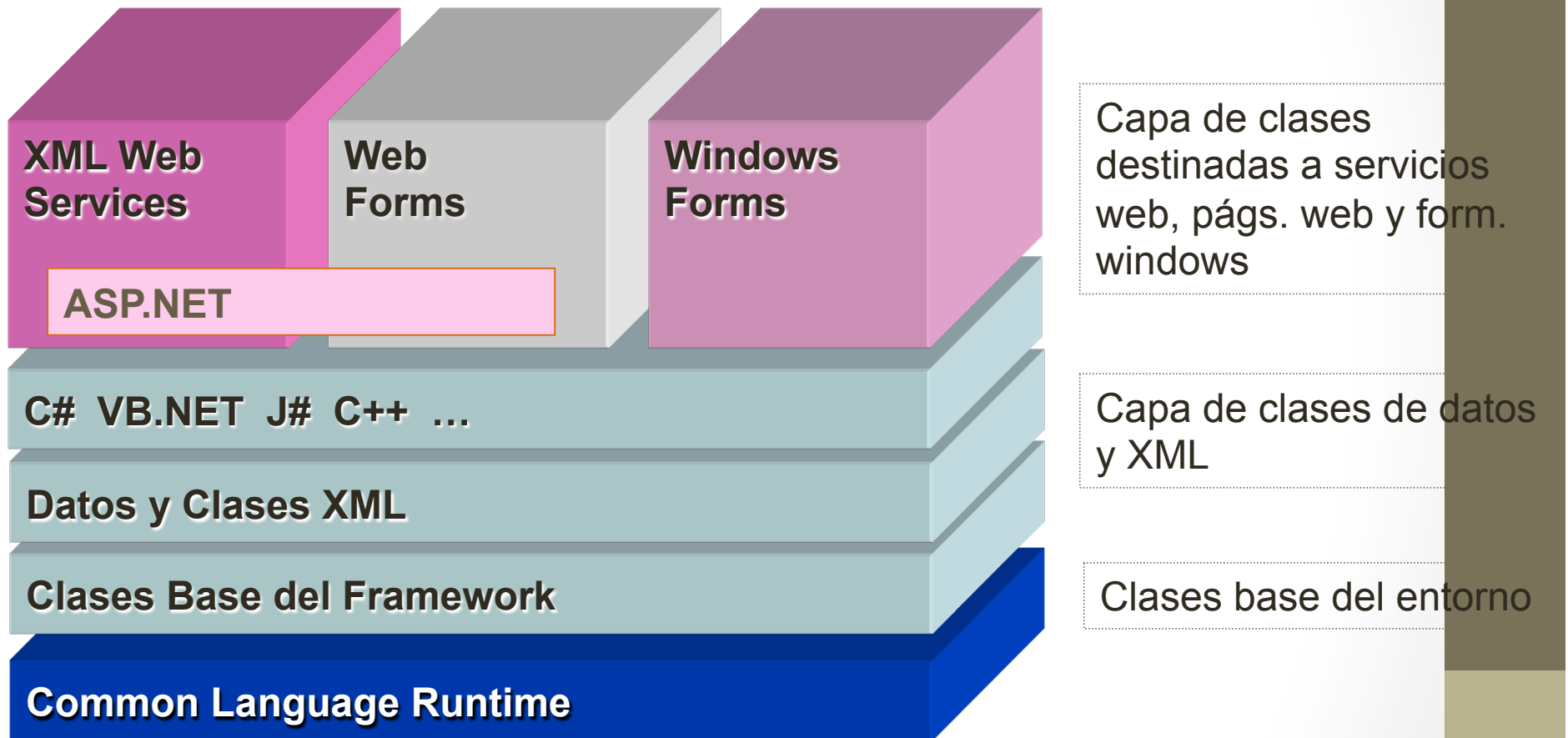
Tecnología dependiente del servidor

- La ventaja principal radica en la **seguridad que tiene el programador sobre su código**:
 - Éste se encuentra únicamente en los archivos del servidor que al ser solicitado a través del web, es ejecutado.
 - Los usuarios no tienen acceso más que a la página resultante en su navegador

Qué es ASP.NET?

- Plataforma de para construir **Aplicaciones Web y Servicios Web** que funcionan bajo IIS.
- Parte del Framework .NET
- Tecnología del lado del servidor
- Lenguajes ASP.NET
 - Orientados a Objeto
 - Dirigidos por Eventos
 - Compilados en el Servidor
- Soporte de múltiples lenguajes:
 - C#
 - VB.NET
 - Jscript.NET
 - J#

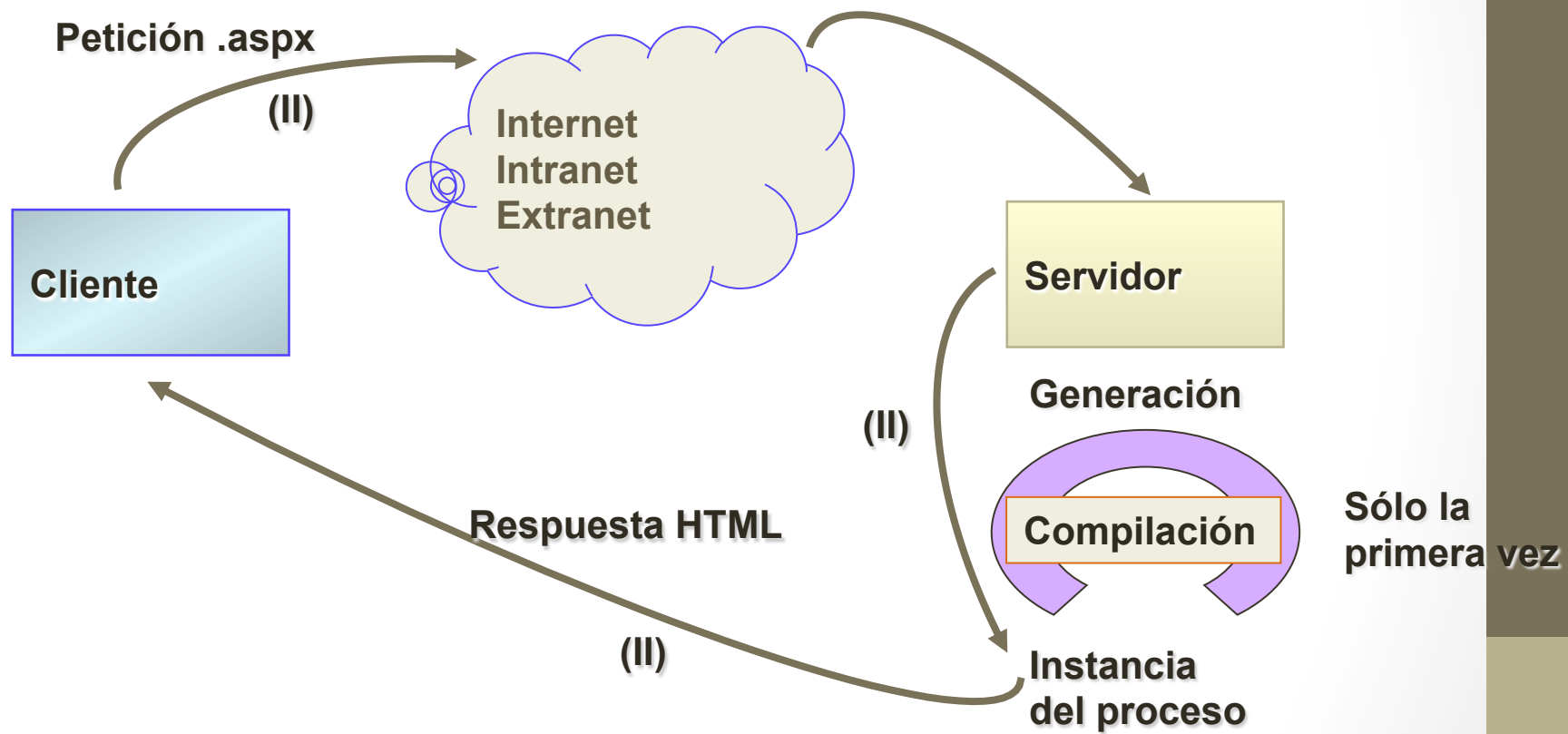
Framework de Microsoft .NET



“Ejecución”

¿Cómo funciona?

- **Llamada a una página ASP .NET**



Aplicaciones Web ASP.NET

- Combinación de archivos, páginas, manejadores, módulos y código ejecutable que puede invocarse desde un **directorio virtual**.
- Se dividen en varias páginas web.
- Comparten un conjunto de recursos y opciones de configuración común.
- Cada aplicación tiene su propio:
 - Conjunto de caché
 - Datos de estado de sesión

Dónde se guarda la aplicación??

→ Directorio Virtual

- Una aplicación web sólo existe en una localización que ha sido publicada por IIS como un **Directorio Virtual**.
- Un **directorio virtual** es un recurso compartido identificado por un alias que representa la localización física en el servidor.
- `//localhost` es la carpeta virtual raíz del ordenador (`\InetPub\wwwroot`)

Directorio virtual

- Directorio virtual: estructura de agrupación básica que delimita una aplicación.
- Creación y administración desde IIS (Internet Information Server)



Formularios Web (I)

- Técnicas para Rápido Desarrollo de Aplicaciones (RAD).
- Podemos:
 - Arrastrar y soltar controles en un formulario
 - Escribir el código soporte
- La aplicación se desarrolla para un servidor web.
- Los usuarios interactúan con la aplicación a través de un navegador.

Formularios Web (II)

- Proporcionan una aproximación orientada a:
 - objetos
 - eventos
 - gestión de estado
- Programación del lado del servidor para manejar eventos del lado del cliente:
 - pueden ejecutarse, virtualmente, sobre cualquier navegador compatible con HTML.

Formularios Web (III)

- Todos los controles de servidor deben aparecer dentro de una etiqueta `<form>`, y esta etiqueta tiene que contener el atributo `runat="server"`.
- Este atributo indica que el formulario se debe procesar en el servidor.
- También indica que los controles que contiene pueden ser accedidos por scripts del servidor:
`<form runat="server">`
 ...HTML + controles de servidor
`</form>`
- **Una página .aspx debe contener un único control `<form runat="server">`**

En VisualStudio...

Sitio Web o Proyecto Web?

- Sitio Web: conjunto de páginas Web independientes.
 - Para páginas Web sencillas (ej, página Web personal...)
- **Proyecto Web:** conjunto de páginas Web enlazadas con un archivo de proyecto.
 - Para aplicaciones avanzadas
 - Podemos referenciar DLLs, etc

Necesitamos Internet Information Server?

- Visual Studio dispone de su propio servidor de desarrollo, por lo que para hacer pruebas en nuestro ordenador no necesitamos tener instalado IIS.
- Sin embargo, para poder desplegar nuestra aplicación en un servidor, si lo necesitaríamos.

En Visual Studio...

Dónde se guarda la aplicación?

- **File system**
- C:\Documents and Settings\aaaa\Mis documentos
\Visual Studio 2005\WebSites\WebSite2
 - <http://localhost:3371/WebSite2/Default.aspx>
- **Local IIS:** Carpeta del sitio web por defecto (<http://localhost>)
 - Por ejemplo C:\Inetpub\wwwroot\WebSite1

Code-inline vs Code-behind

- ▶ “Etiquetas” declarativas
 - HTML, controles de servidor, texto estático
- ▶ A diferencia de ASP, buena separación entre el código y las etiquetas



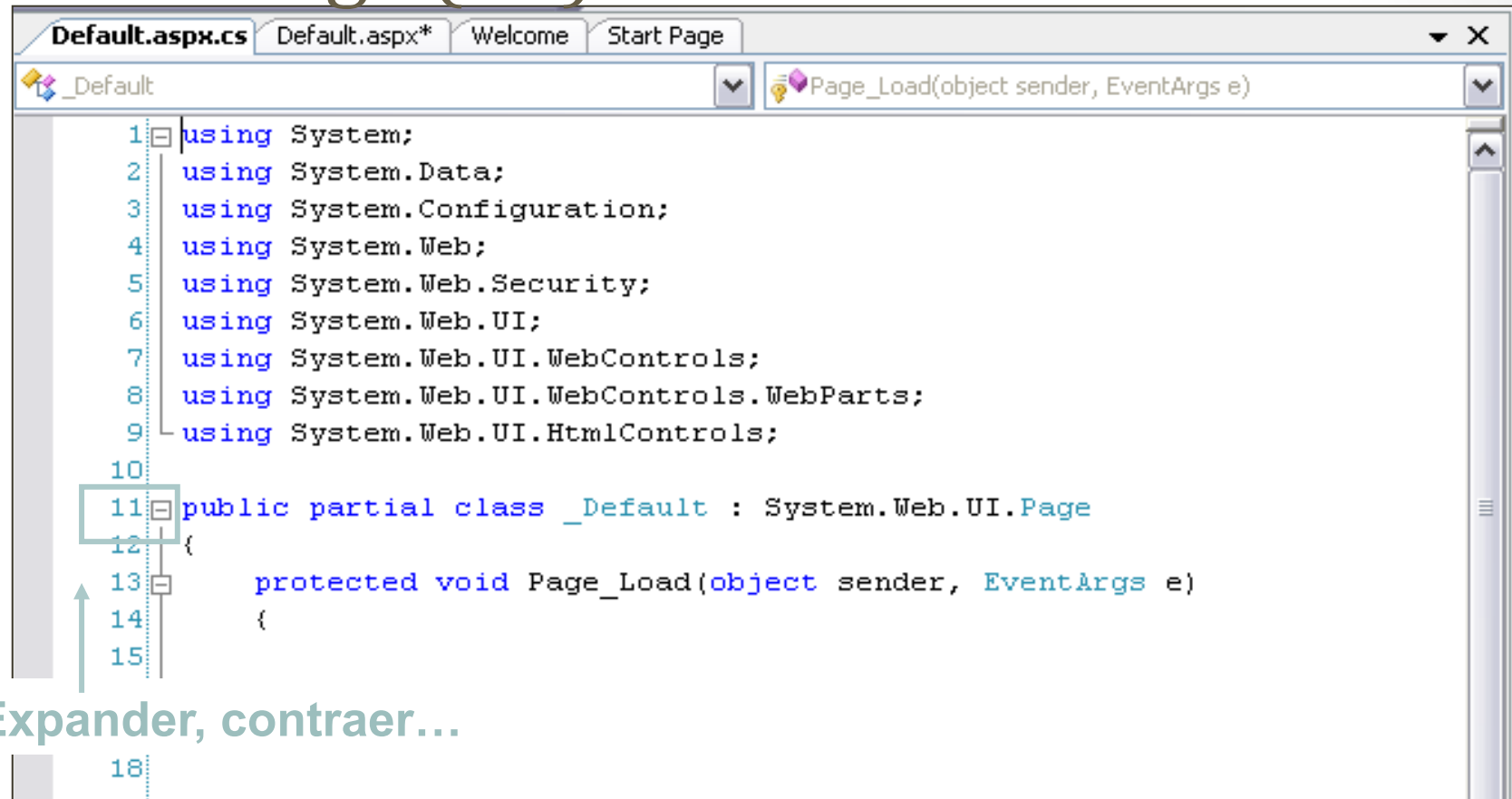
Code-behind

- El código para manejar eventos se sitúa en un **archivo físico separado** de la página que contiene los controles de servidor y las etiquetas.
 - Extensión *.aspx*
 - Extensión *.cs*, *.vb* ... (*code-behind*)
- UTIL mantenerlo por separado :

Es común en grupos de proyectos tener

 - diseñadores trabajando en la IU de la aplicación
 - y desarrolladores en el comportamiento o código.

Ver código (C#)



```
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Web;
5 using System.Web.Security;
6 using System.Web.UI;
7 using System.Web.UI.WebControls;
8 using System.Web.UI.WebControls.WebParts;
9 using System.Web.UI.HtmlControls;
10
11 public partial class _Default : System.Web.UI.Page
12 {
13     protected void Page_Load(object sender, EventArgs e)
14     {
15
16
17
18
```

Expander, contraer...

Definición de una clase parcial

Procedimiento que responde al evento LOAD de la página

Cuadro de herra... Web.config

Estándar

- Puntero
- Label
- TextBox
- Button
- LinkButton
- ImageButton
- HyperLink
- DropDownList
- ListBox
- CheckBox
- CheckBoxList
- RadioButton
- RadioButtonList
- Image
- ImageMap
- Table
- BulletedList
- HiddenField
- Literal
- Calendar

Web.config Default.aspx* Default.aspx* Página de inicio Ex

Propiedades

Button1 System.Web.UI.WebControls.Button

(Expressions)	
(ID)	Button1
AccessKey	
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CausesValidation	True
CommandArgument	
CommandName	
CssClass	
Enabled	True
EnableTheming	True
EnableViewState	True
Font	
ForeColor	
Height	
OnClientClick	
PostBackUrl	
SkinID	
TabIndex	0
Text	Button
ToolTip	
UseSubmitBehavior	True
ValidationGroup	
Visible	True

Default

```
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.
{
    protected void Page_Load(object sender,
    {
        this.
    }
}
```

AppRelativeTemplateSourceDirectory
AppRelativeVirtualPath
AsyncTimeout
BuildProfileTree
Button1
Cache
ChildControlsCreated
ClearChildControlState
ClearChildState
ClearChildViewState

Lista de errores 0 errores 0 advertencias 0 mensajes

Descripción

Lista de errores Resultados

Listo

Indice

1. Páginas maestras
2. Controles de servidor
3. Eventos
4. Maquetación con CSS
5. Navegación entre WebForms

1

Páginas
maestras

Página maestra

- Las páginas maestras permiten crear un diseño coherente para las páginas de la aplicación.
- Se puede definir el aspecto, el diseño y el comportamiento estándar que desea que tengan todas las páginas (o un grupo de páginas) de la aplicación en una sola página maestra.
- A continuación, se crean páginas de contenido individuales que incluyan el contenido que desea mostrar.

Al solicitar una página...

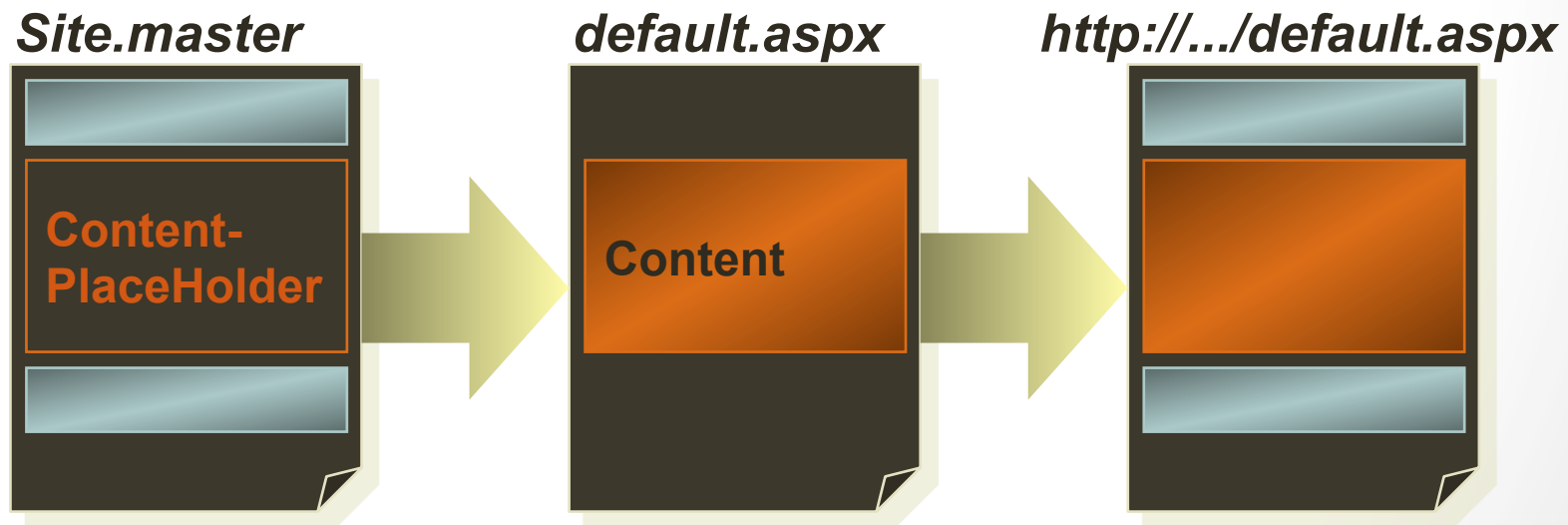
- Cuando los usuarios solicitan las páginas de contenido, éstas son combinadas con la página maestra con el fin de generar una salida que combine el diseño de la página maestra con el de la página de contenido.

Ventajas

- Realización de cambios de diseño en una sola ubicación; los cambios se verán reflejados en todas las páginas que usan la página maestra.
- Reutilización de la interfaz de usuario
- Mejor experiencia del usuario final: páginas más coherentes

Cómo funciona...

- Master pages definen el contenido común y los contenedores de contenido (content placeholders)
- Content pages hacen referencia a las paginas maestras y llenan a los contenedores con su contenido.



En tiempo de ejecución

- IIS controla las páginas maestras en la secuencia siguiente:
 - Los usuarios solicitan una página escribiendo la dirección URL de la página de contenido.
 - Cuando se captura la página, se lee la **directiva @ Page**. Si la directiva hace referencia a una página maestra, también se lee la página maestra. Si las páginas se solicitan por primera vez, se compilan las dos páginas.
 - La página maestra con el contenido actualizado se combina en el árbol de control de la página de contenido.
 - El contenido de los controles Content individuales se combina en el control **contentplaceholder** correspondiente de la página maestra.
 - La página combinada resultante se representa en el explorador.

Page.Master

- Nueva propiedad (Master) de System.Web.UI.Page
- Esta propiedad contiene una referencia a la página maestra de la página de contenido, por tanto provee a una página contenido de acceso programático a la página maestra
 - Determina si la pagina tiene asociada una maestra
 - Acceso a los controles definidos en la maestra pudiendo escribir código soporte en las páginas de contenido
 - Acceso a métodos públicos y propiedades definidas en la maestra
- Integración a nivel código de las páginas maestras y contenidos

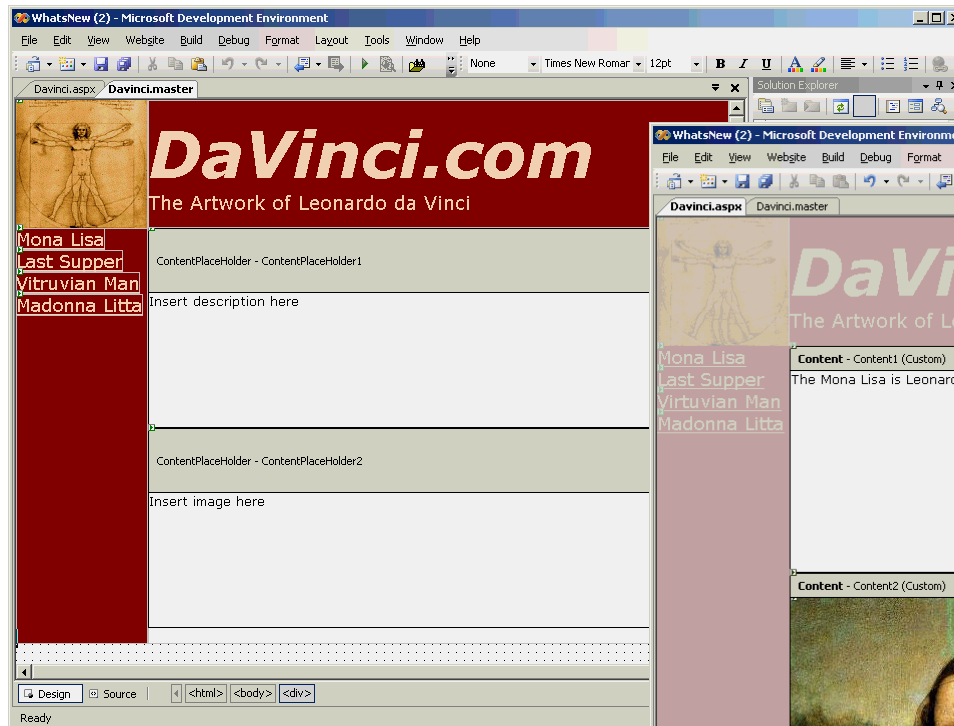
Desde el navegador..

- Desde la perspectiva del usuario, la combinación de las páginas maestras y las páginas de contenido da como resultado una única página. La dirección URL de esta página es la de la página de contenido.

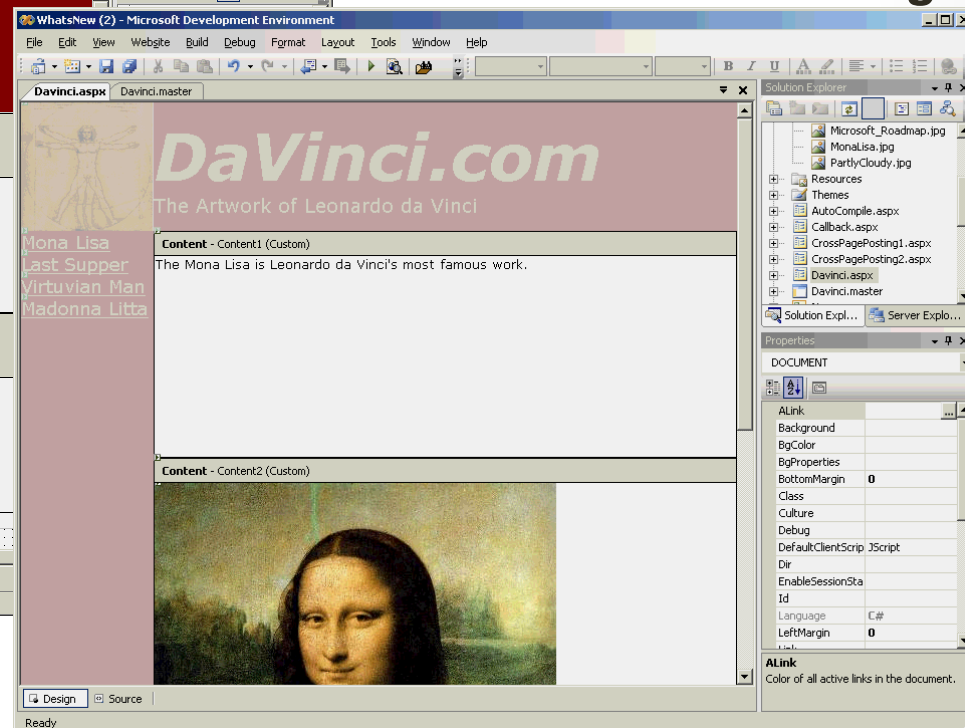
En Visual Studio...

- Herencia visual

Master Page



Content Page



2

Controles de servidor

Los controles de servidor...

- Tienen propiedades que pueden ser establecidas
 - declarativamente (en la etiqueta)
 - o mediante programación (en el código).
- Junto con la página, **tienen eventos** que los desarrolladores pueden manejar
 - para ejecutar acciones específicas durante la ejecución de la página
 - o en respuesta a una acción del lado del cliente que envía la página al servidor.

Controles de servidor

- Soporte para características avanzadas: enlace de datos, plantillas..
- Se emplea el prefijo **asp:** junto con el atributo **runat="server"**.
`<asp:TextBox id="text" runat="server" />`

Tipos de controles ASP.NET

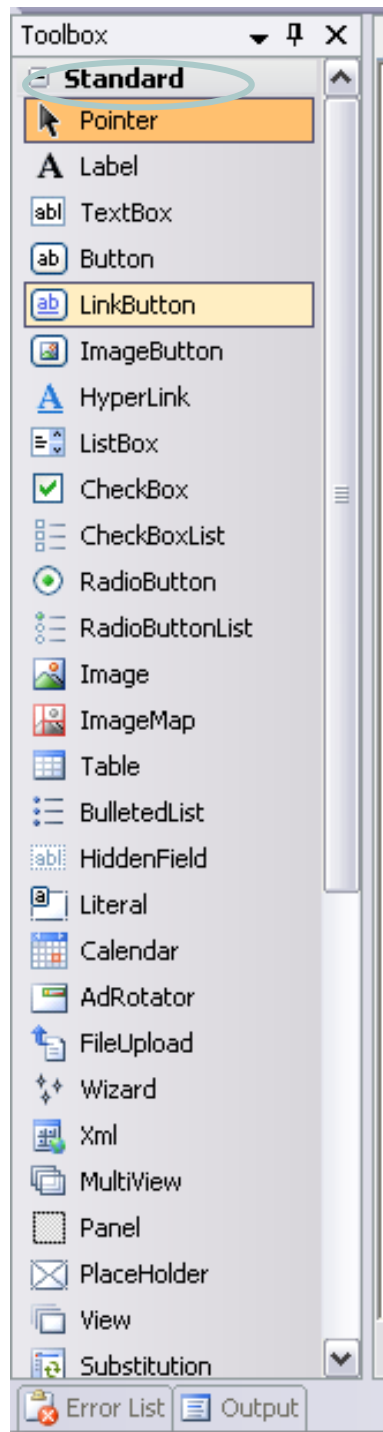
- 1. Controles HTML
- 2. Controles estándar (web)
- 3. Controles de validación
- 4. Controles de login
- 5. Controles de navegación
- 6. Controles Webparts
- 7. Controles de datos
- 8. Controles de usuario

Principales controles web (estándar)

Label	Se utiliza para mostrar texto dinámico (cambiamos sus propiedades a través del código del servidor)
Hyperlink	Muestra un enlace a otra página.
TextBox	Permite introducir texto al usuario. Propiedad Textmode valores: Single, Multiline o Password.
Image	Sirve para mostrar una imagen en la página web.
Button	Se usa en un Formulario web para crear un control de tipo submit (evento OnClick) o un control de comando tipo botón (evento OnCommand).
LinkButton	Apariencia de hipervínculo pero funciones de control de un botón (envío formulario)
ImageButton	Muestra una imagen que maneja eventos tipo click.
Checkbox	Sirven para añadir casillas de verificación a una página.
RadioButton	Crea un botón de radio individual en la página.

Principales controles web

DropDownList	Proporciona un buen método para que los usuarios elijan elementos de una lista en un espacio pequeño. Cada elemento se crea con un control ListItem.
ListBox	Propiedad SelectionMode: Single, Multiple. Cada elemento se crea con un control ListItem.
CheckBoxList	Permite presentar una lista de opciones pudiendo el usuario seleccionar varias.
RadioButtonList	Permite crear una lista de botones de radio de opciones excluyentes.
Panel	Puede usarse como contenedor de otros controles.
Table, TableRow, TableCell	Permiten crear dinámicamente una tabla mediante programación.



Controles estándar

- AdRotator
- Placeholder
- ImageMap
- BulletedList
- HiddenField
- FileUpload
- Wizard
- Xml
- MultiView
- Substitution...

TextBox

Página.aspx

```
<form id="form1" runat="server">
<div>
<p>
    Username: <asp:TextBox id="userTextBox"
        TextMode="SingleLine"
        Columns="30" runat="server" />
</p>
<p>
    Password: <asp:TextBox id="passwordTextBox"
        TextMode="Password" Columns="30"
        runat="server" />
</p>
<p>
    Comments: <asp:TextBox id="commentsTextBox"
        TextMode="MultiLine" Columns="30" Rows="10"
        runat="server" />
</p>
</div>
</form>
```

Username:

Password:

Comments:

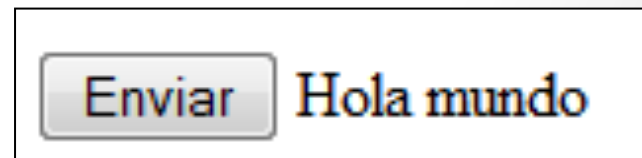
Button

Página.aspx

```
<form id="form1" runat="server">  
<asp:Button id="BotonEnviar" Text="Enviar"  
    runat="server" OnClick="WriteText" />  
<asp:Label id="Label1" runat="server" />  
</form>
```

Página.aspx.cs

```
protected void WriteText(object sender, EventArgs  
    e)  
{  
    Label1.Text = "Hola mundo";  
}
```



ImageButton

Página.aspx

```
<form id="form1" runat="server">
  <div>
    <asp:ImageButton id="BotonImagen" ImageUrl="~/
    garfield.gif"
    runat="server" OnClick="WriteText" />
    <asp:Label id="Label4" runat="server" /> </form>
```

Página.aspx.cs

```
protected void WriteText(object sender, ImageClickEventArgs e)
{
    Label4.Text = "Coordenadas:" + e.X + "," + e.Y;
}
```



Panel

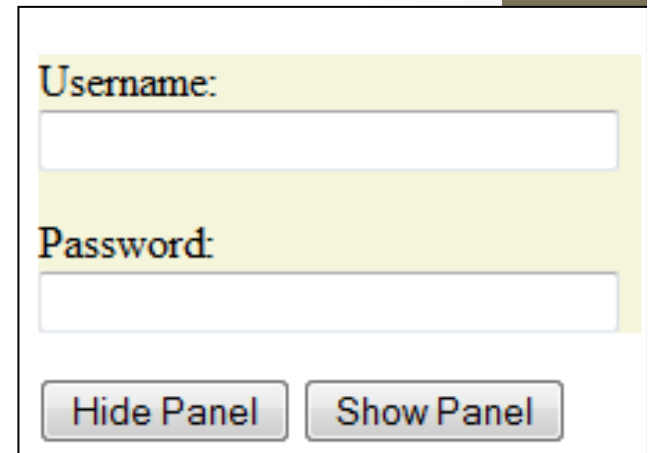
Página.aspx

```
<form id="form1" runat="server">
  <asp:Panel id="myPanel" BackColor="Beige" Width="220" runat="server">
    <p>Username: <asp:TextBox id="usernameTextBox" Columns="30"
      runat="server" /></p>
    <p>Password: <asp:TextBox id="TextBox1" TextMode="Password"
      Columns="30" runat="server" /></p>
  </asp:Panel>
  <asp:Button id="hideButton" Text="Hide Panel" OnClick="HidePanel"
    runat="server" />
  <asp:Button id="showButton" Text="Show Panel" OnClick="ShowPanel"
    runat="server" />
</form>
```

Página.aspx.cs

```
protected void HidePanel(object sender, EventArgs e)
{ myPanel.Visible = false; }
```

```
protected void ShowPanel(object sender, EventArgs e)
{ myPanel.Visible = true; }
```



The screenshot shows a web application interface. At the top, there is a light yellow rectangular panel. Inside this panel, the text "Username:" is followed by a text input field. Below that, the text "Password:" is followed by a password input field (indicated by a small square icon). At the bottom of the yellow panel, there are two buttons: "Hide Panel" and "Show Panel". The "Show Panel" button is currently visible, indicating the panel is in its default state.

3

Eventos de los controles de servidor

Modelo de eventos

- En las páginas Web ASP.NET, los eventos asociados a los controles de servidor se originan en el cliente (explorador) pero los controla la página ASP.NET en el servidor Web.
- La información del evento se captura en el cliente y se transmite un mensaje de evento al servidor mediante un envío HTTP.
- La página debe interpretar el envío para determinar el evento ocurrido y, a continuación, llamar al método apropiado del código del servidor para controlar dicho evento.

Enlazar eventos a métodos

- Un evento es un mensaje que envía un objeto cuando ocurre una acción (Ej. "se ha hecho clic en un botón") . La acción puede estar causada por la interacción del usuario, como un clic, o por otra lógica del programa.
- En una aplicación, se debe traducir el mensaje en una llamada a un método del código.
- El enlace entre el mensaje del evento y un método específico (es decir, un controlador de evento) se lleva a cabo utilizando **un delegado de eventos**.

Eventos y delegados

- El objeto que provoca el evento se conoce como remitente del evento. El objeto que captura el evento y responde a él se denomina receptor del evento.
- En las comunicaciones de eventos, el remitente del evento no sabe qué objeto o método recibirá los eventos que provoca. **Se necesita un intermediario (o mecanismo de tipo puntero) entre el origen y el receptor.**
- .NET Framework define un tipo especial (**Delegate**) que proporciona la funcionalidad de un **puntero a función**.

Eventos y delegados

- Un delegado es una clase que puede guardar una referencia a un método.
- A diferencia de otras clases, una clase de delegado tiene un prototipo y puede guardar referencias únicamente a los métodos que **coinciden con su prototipo**.
- Por lo tanto, **un delegado equivale a un puntero a función con seguridad**.
- Por convención, los delegados de evento de .NET Framework tienen dos parámetros, el origen que provocó el evento y los datos del evento

Eventos y delegados

- Los delegados de evento personalizados sólo son necesarios cuando un evento genera datos de evento. Muchos eventos, incluidos algunos eventos de interfaz de usuario, como los clics, no generan datos de evento.
- En estos casos, es apropiado el delegado proporcionado en la biblioteca de clases para el evento sin datos, [System.EventHandler](#).

`delegate void EventHandler(object sender, EventArgs e);`

- Los delegados de evento son de multidifusión, lo que significa que pueden guardar referencias a más de un método de control de eventos.
- En las páginas Web ASP.NET, no es necesario codificar explícitamente delegados si el control se crea mediante declaración (en el marcado) en la página.

Manejadores de eventos

- El prototipo de los métodos manejadores de eventos deben coincidir con el prototipo del *delegado EventHandler*.
- Por tanto, los manejadores de evento en ASPnet devuelven *void* y tienen **dos parámetros**:
 - *Objeto que lanza el evento*
 - *Argumentos del evento*: información específica del evento (*EventArgs* o tipo derivado)
- Por ejemplo, para un control ImageButton de servidor Web, el segundo argumento es de tipo ImageClickEventArgs, que incluye información sobre las coordenadas donde el usuario ha hecho clic..

Asociar el manejador al control

- Escribirlo manualmente

Archivo Default.aspx

```
<asp:Button ID="Button1" runat="server"
  onclick="EventoClick" Text="Button" />
```

Archivo Default.aspx.cs

```
protected void EventoClick(object sender,
  EventArgs e)
{ }
```

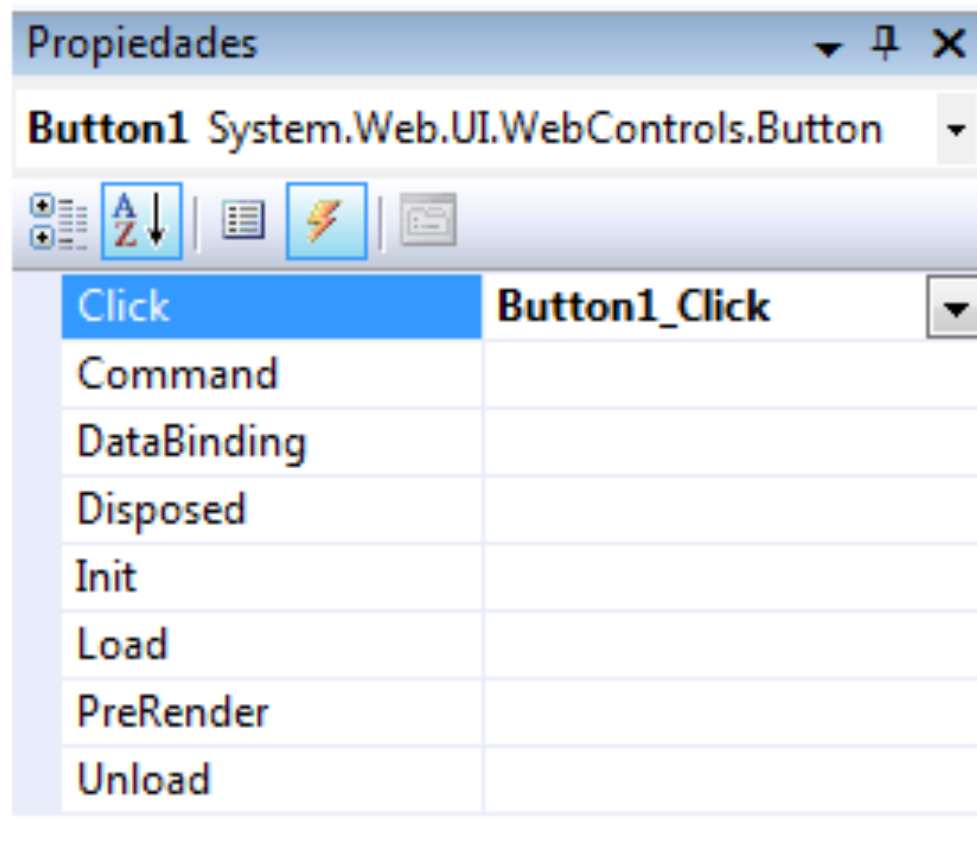
- Pulsar doble click sobre un control en vista diseño (evento por defecto)

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

Al compilar la página, ASP.NET busca un método denominado EventoClick y confirma que éste tiene la firma adecuada (acepta dos argumentos, uno de tipo **Object** y otro de tipo EventArgs). A continuación, ASP.NET enlaza automáticamente el evento al método

Asociar el manejador al control

- Seleccionar el evento de la ventana de propiedades del control (y asociar un nombre de método manejador del evento o doble click)



Para crear un controlador de eventos en tiempo de ejecución con Visual C#

- Normalmente esto se hace cuando se están creando controles mediante programación.
- Para ello **se ha de crear una instancia del delegado EventHandler**, a la que se debe pasar la dirección del método al que se va a enlazar.
- Después es necesario agregar el objeto delegado a la lista de métodos a los que se llama cuando se produce el evento.
- En el ejemplo de código siguiente se muestra la manera de enlazar el evento **Click** del control Button1 a un método denominado myEventHandler:

```
Button1.Click += new System.EventHandler  
(this.myEventHandler);
```

Ejemplo

```
Button b = new Button;  
b.Text = "Click";  
b.Click += new System.EventHandler(ButtonClick);  
Placeholder1.Controls.Add(b);
```

Tipos de eventos

- **Eventos de envío (POSTBACK).** Son los eventos lanzados por controles que emiten un envío (post) al servidor de manera inmediata para procesar el Web Form.
 - Por ejemplo al enviar un formulario pulsando sobre el botón de aceptación. Estos controles son: Button, Link Button e Image Button.
- **Eventos de caché (NO-POSTBACK).** Estos eventos se producen en la vista y serán procesados en el servidor cuando se envíe la información mediante un evento de envío.
 - Por ejemplo el seleccionar un elemento de una lista provoca un cambio de estado de la misma que luego, en servidor, podremos obtener. Controles con eventos de caché son TextBox, DropDownList, CheckBox...

Eventos postback vs no-postback

- Una página se carga después de cada petición: fenómeno conocido como **PostBack (=envío)**.
- Eventos **PostBack**:
 - causan que la información del formulario se envíe al servidor inmediatamente.
- Eventos **no-PostBack (o cached)**:
 - la información se envía en el siguiente evento **postback**.
 - Los eventos se guardan en una cola en el cliente hasta que un evento **postback** ocurre.

Eventos postback vs no-postback (II)

- Button, Link Button y Image Button causan eventos **postback**.
- TextBox, DropDownList, ListBox, RadioButton y CheckBox, proveen eventos **cached**.
 - Sin embargo podemos sobrecargar este comportamiento en los controles para poder realizar eventos postback, cambiando la propiedad **AutoPostBack a true**.

Conectar varios eventos con un único controlador de eventos

```
<asp:Button ID="Button1" onclick="Button_Click" runat="server"
  Text="Button1" />
<br />
<asp:Button ID="Button2" onclick="Button_Click" runat="server"
  Text="Button2" />
```

- **Para determinar el control que provocó el evento**

```
private void Button_Click(object sender, System.EventArgs e)
{
    Button b = (Button) sender;
    Label1.Text = b.ID;
}
```

se muestra el controlador del evento **Click** de un control **Button** al que llaman varios botones diferentes. El controlador muestra la propiedad **ID** del botón que provocó el evento.

Eventos de página

- Las páginas ASP.NET provocan eventos de ciclos de vida como **Init**, **Load**, **PreRender** y otros.
- De manera predeterminada, los eventos de página se pueden enlazar a los métodos utilizando la convención de nomenclatura **Page_nombreDeEvento**.
 - Por ejemplo, con el fin de crear un controlador para el evento **Load** de la página, se puede crear un método denominado **Page_Load**.
 - En tiempo de compilación, ASP.NET buscará los métodos que se basen en esta convención de nomenclatura y realizará el enlace automáticamente entre el evento y el método.
- Se puede utilizar la convención **Page_nombreDeEvento** para cualquier evento expuesto por la clase **Page**.

Eventos de página

- Las páginas ASP.NET enlazan automáticamente los eventos de páginas a los métodos que tienen el nombre **Page_evento**.
- Este enlace automático lo configura el atributo **AutoEventWireup** de la directiva @ Page, cuyo valor se establece de forma predeterminada en **true**.
- Si establece **AutoEventWireup** en **false**, la página no busca automáticamente los métodos que usen la convención de nomenclatura **Page_evento**. Por tanto, se pueden crear los métodos con cualquier nombre y enlazarlos a los eventos de páginas explícitamente.

Propiedad IsPostBack

- Una página se carga después de cada petición: fenómeno conocido como **PostBack (=envío)**.
- El evento Page_Load se produce en cada petición.
- **Page.IsPostBack** para ejecutar código condicional

```
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        Response.Write("<br>Page has been posted back.");
    }
}
```

Importante: Los métodos de control de eventos de página no requieren ningún argumento. Estos eventos (como Load) pueden aceptar los dos argumentos estándar, pero en estos argumentos no se pasa ningún valor.

4

Maquetación con CSS

¿Qué es CSS?

- CSS es una herramienta que permite definir la presentación de un documento
- Permite crear un conjunto de estilos en una única localización
- Las páginas a las que se aplica la misma hoja de estilos tendrán las mismas fuentes, colores y tamaño
- Proporcionan una estética homogénea en todas las páginas de un sitio web

Tipos de hojas de estilos

- Hoja de estilo externa
 - Se emplazan todas las reglas de estilo en una única hoja de estilos externa
 - Se enlaza esta hoja de estilos externa con todas las páginas que vayan a tener la misma apariencia
- En un formulario web:
 - `<link rel="Stylesheet" type="text/css" href="~/hoja.css" />`

Se sitúa dentro del elemento HEAD

Tipos de hojas de estilos

- Hoja de estilo interna
 - Hoja de estilo incrustada dentro de un documento
 - Se definen las reglas de estilo entre las etiquetas:

```
<style type="text/css">  
a { background-color: #ff9;  
    color: #00f; }  
</style>
```

- Problema: Se debe reescribir en cada página

Se sitúa dentro del elemento HEAD

Tipos de hojas de estilos

- Hoja de estilo en línea
 - Permiten asignar un estilo a un elemento determinado, usando el atributo **style**

```
<a href="/" style="background-color: #ff9;  
color: #00f;"> Home</a>
```

Selectores de estilos

- En caso de utilizar hojas externas o internas cada regla de estilo tiene un selector
- Un selector es el tipo de elemento al que se va a aplicar el estilo
- Ejemplo:
a {
 background-color: #ff9;
 color: #00f;
}
- En ASP.NET hay dos tipos de selectores:
 - Tipos de elementos y clases

Selector de tipo elemento

- El estilo se aplica a todos los elementos del mismo tipo

```
h2 { color: #369;}
```

- Cambia de color todas las cabeceras de segundo nivel

Selector de tipo clase

- Se utiliza cuando asignamos a cada elemento una clase determinada

```
<p class="pageinfo">
```

```
    Copyright 2010
```

```
</p>
```

- La hoja de estilos contendrá para cada clase una serie de características

```
.pageinfo
```

```
{
```

```
    font-family: Arial;
```

```
    font-size: x-small;
```

```
}
```

Los selectores de clase van precedidos por un .

Propiedades de estilo

- **Font:** Tipo de fuente, tamaño, color...
- **Background:** color de fondo, imagen de fondo...
- **Block:** espacio entre párrafos, líneas, palabras...
- **Box:** personalizar tablas, colores, bordes...
- **Border:** dibuja bordes alrededor de diferentes elementos
- ...

CssClass

- Asociar selector de tipo clase en Formularios Web en ASP.NET:

```
<head runat="server">
```

```
<title>Probando CSS</title>
```

```
<link rel="Stylesheet" type="text/css" href="hoja.css" />
```

```
</head>
```

```
<asp:TextBox ID="TextBox1" CssClass="textbox"  
runat="server" />
```

hoja.css

```
.textbox { font-family:  
Arial; background-color:  
#0099FF; border: 1px  
solid }
```

Maquetación con CSS

- ***maquetar una pagina web es pasar el diseño a código HTML***, poniendo cada cosa en su lugar (una cabecera, un menu, etc.).
- Hasta hace unos años la única manera de maquetar una pagina web era mediante **tablas HTML** (<table>), pero esto tiene muchas desventajas y limitaciones
 - el uso de las tablas está condicionado a la mera tabulación de datos,
 - Un diseño con tablas no es flexible, es decir, que no podemos cambiar la distribución de los elementos en la página, a menos que la volvamos a hacer.
 - Cada Explorador renderiza de manera distinta cada documento HTML y con estructuras con tablas el cambio es más notorio
 - Ocupa más espacio y más ancho de banda.
 - Google no indexa de igual manera las páginas con estructuras basadas en tablas.
- Por eso la técnica de maquetación fue evolucionando con los años hasta llegar al punto donde no se usan tablas, si no capas (**los famosos DIVs**) a las que se le dan formato mediante **CSS**.

Divs

- Las capas, layouts o divs son la misma cosa con distinto nombre, para tener un concepto mental de lo que son, podemos imaginarlos como ***contenedores o bloques donde podemos meter lo que queramos dentro*** (imágenes, texto, animaciones, otro bloque, o todo al mismo tiempo) a los que se le asigna un ancho, alto y posición, de esta manera se van a ir posicionando consiguiendo la estructura que queremos.

Formato a un DIV

- Para darle formato a un DIV tenemos que identificarlo de alguna forma, para esto existe **el atributo ID**, en el pondremos el nombre del DIV para luego llamarlo desde la hoja de estilos, la forma de escribirlo es así:

- `<div id="capa1">¡Esta es mi primer capa!</div>`

```
#capa1{  
  width:210px;  
  height:300px;  
  background-color:green;  
}
```

¡Esta es mi primer capa!

Class o id?

- La diferencia entre una clase (.) y un bloque (#) es que el bloque es único e irrepetible en la pagina, es decir, si creamos un estilo de bloque **"#busqueda"** para mostrar el cuadro de búsqueda no podremos usarlo otra vez en la misma pagina, en cambio si a **"#busqueda"** lo convertimos en una clase **".busqueda"** podremos usarlo cuantas veces queramos.

Propiedades de maquetación

```
<div style = "display: table; margin-left: auto; margin-right: auto;
    width: 500px;">
  <div style = "height: 45px; width: 500px;"></div>
  <div style = "float: left; height: 75px; width: 150px;"></div>
  <div style = "float: right; height: 75px; width: 350px;"></div>
  <div style = "clear: both; height: 35px; width: 500px;"></div>
</div>
```

- Los elementos DIV pueden centrarse utilizando los atributos **margin-left: auto; margin-right: auto;**
- La **propiedad float** ajusta los elementos hacia el margen indicado (cuando tenemos capas adyacentes).
- A veces necesitarás tener una capa o bloque que no tenga capas a su/s lados, para eso esta la propiedad CSS **Clear**. (ej el pie de página). Esta propiedad se utiliza en conjunto con *float* y **sirve para evitar que una capa se posicione a cualquiera de los lados**

Cuidado

- En las estructuras clásicas con tablas, podíamos utilizar tamaños en porcentaje. Aunque aquí, también podemos utilizar porcentajes, el dibujado de la página puede no verse como se esperaba.

Maquetar con CSS links

- <http://www.comocrearunsitioweb.com/conceptos-basicos-css>
- <http://www.webexperto.com/articulos/art/232/por-que-maquetar-con-estandares/>
- <http://www.desarrolloweb.com/manuales/manual-css-hojas-de-estilo.html>
- <http://www.comocrearunsitioweb.com/maquetando-pagina-web-css>

5

Navegación entre formularios

Navegación tradicional

- En HTML:
 - Elemento `<a>` y atributo `href`
 - `Regístrate aquí`
 - La página `Login.aspx` debe estar en el mismo directorio que la página que contiene el enlace
 - Después de pulsar sobre el enlace se muestra la página `Login.aspx`
- En ASPx:
 - `<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="~/Login.aspx">Regístrate aquí</asp:HyperLink>`

Diferentes formas de navegar

- **Control hipervínculo**
 - Navega a otra página
- **Método `Response.Redirect`**
 - Navega a otra página por medio del código. (Equivalente a navegar a través de un enlace)

Hipervínculos y Redirección

- Los hipervínculos responden a eventos click mostrando la página especificada en la propiedad `NavigateURL` del control.
- Si quieres capturar un click en el código, debes usar los eventos de un `LinkButton` o `ImageButton` y utilizar `Response.Redirect("SiguientePagina.aspx");`
- También podemos pasar parámetros a la nueva página.

Paso de parámetros a otra página

- Paso de parámetros en la URL.
 - Almacena los datos en la colección QueryString
 - Múltiples parámetros separados por &

```
int valor = 22;
```

```
int valor1 = 25;
```

```
Response.Redirect("Default4.aspx?par1=" + valor);
```

```
Response.Redirect("Default4.aspx?par1=" + valor +  
"&par2=" + valor1);
```

```
http://localhost:49999/Default4.aspx?  
par1=22&par2=25
```


Paso de parámetros a otra página

- Limitaciones con QueryString
 - Los caracteres utilizados deben ser caracteres permitidos en una URL
 - La información es visible a los usuarios en la barra del navegador
 - Los usuarios pueden modificar la información provocando errores inesperados
 - Muchos navegadores imponen un límite en la longitud de la URL

Paso de parámetros a otra página

- Caracteres especiales:
 - & (para separar múltiple query strings)
 - + (alternativa para representar un espacio)
 - # (especifica un marcador en una página web)
- Solución:
 - Utilizar los métodos de la clase HttpServerUtility para codificar los datos

```
Response.Redirect("WebForm2.aspx?par1="+  
Server.UrlEncode(" &hola "));
```

Paso de parámetros a otra página

- El método `UrlEncode` reemplaza los caracteres especiales por secuencias de escape

`http://localhost:49999/WebForm2.aspx?par1=+%26hola+`

- La recuperación de datos codificados mediante el método `UrlEncode` con `QueryString` es automática en ASP.NET (no es necesario utilizar un método que decodifique los datos)

Objeto Request

- **Request:** proporciona información sobre la petición HTTP del cliente que ha provocado la carga de la página actual.
 - Información sobre el cliente (`Request.Browser`, `Request.Browser.IsMobileDevice`, `Request.Browser.Id`)
 - Cookies (`Request.Browser.Cookies`)
 - Parámetros pasados a la página:

```
if (Request.QueryString["par1"] != "")  
    Label1.Text = Request.QueryString["par1"];
```