maestros del web

ASP.NET

Guía de desarrollo de sitios y aplicaciones web dinámicas

CATEGORIA
PROGRAMACIÓN

NIVEL
INTERMEDIO

Fernando Giardina

Es de origen argentino, inició su carrera desarrollando en lenguaje C y Oracle programador IVRs con experiencia en la automatización de procesos y sistemas de CRM. Además, se especializó en Java, Perl, PHP y .NET para el desarrollo de aplicaciones web.

Guía de ASP.NET:
Desarrollo de sitios y aplicaciones web dinámicas

Versión 1 / abril 2011

Nivel: Intermedio

La Guía ASP.NET se encuentra en línea en:
http://www.maestrosdelweb.com/editorial/tutoria-desarrolloweb-asp-net/

Un proyecto de Maestros del Web

- Edición: Stephanie Falla Aroche
- Diseño y diagramación: Iván Mendoza

Contacto

http://www.maestrosdelweb.com/sitio/correo/

Redes sociales

Facebook: http://www.facebook.com/maestrosdelweb
Twitter: http://www.twitter.com/maestros

maestros del web

Indice

Capítulo 1

Introducción a ASP.NET

Capítulo I

Introducción a ASP.NET

ASP.NET es un modelo de desarrollo web unificado creado por Microsoft para el desarrollo de sitios y aplicaciones web dinámicas con un mínimo de código, forma parte de .NET Framework[1] que contiene las librerías necesarias para la codificación. Se pueden usar distintos lenguajes de programación para realizar aplicaciones web en ASP.NET, pero nos vamos a enfocar en el lenguaje C# (C Sharp) el más utilizado para este tipo de desarrollos.

Requisitos para el desarrollo en ASP.NET

- Un editor de código.
- NET Framework.
- Un servidor Web como IIS (Servicios de Internet Information Server).

Adicionalmente a la programación web es necesario tener algún tipo de soporte para el almacenamiento de datos (SQL Server, Oracle, etc.) pero para esta guía vamos a mantener la información en soportes que estén al alcance de todos.

Editor de código

Se requiere de un editor de texto estándar (Notepad, Notepad++) pero existen herramientas con un entorno de desarrollo integrado (IDE) que nos facilita el acceso a las librerías del Framework y nos brinda un entorno amigable para la creación de aplicaciones web en ASP.NET como el Visual Studio.

En el transcurso de la Guía ASP.NET utilizaremos IDE para abordar los ejemplos. Descarga la versión Express[2].

.NET Framework

Es un conjunto de clases que actuarán como soporte de las aplicaciones ASP.NET instaladas en nuestro equipo. Es de distribución gratuita y se puede descargar desde la página de Microsoft. (Incluido al Visual

1 http://www.microsoft.com/net/
2 http://www.microsoft.com/express/Downloads/

Studio .NET).

Servidores Web

Una aplicación ASP.NET además de contar con el .NET Framework instalado necesita de un Servidor Web. Utilizaremos el IIS (Internet Information Server) que viene como complemento de instalación de Windows. Si dentro de nuestros servicios en ejecución no contamos con Internet Information Server debemos agregarlo al equipo donde instalaremos nuestras aplicaciones.

Capítulo 2

Nuestra primera página web

Capítulo 2

Nuestra primera página web

En la introducción de la Guía ASP.NET explicamos los requisitos básicos necesarios para el desarrollo, ahora que cuentas con un equipo IIS, .Net Framework y un entorno de desarrollo procederemos a crear nuestra primera aplicación web en ASP.NET.

Mi primera aplicación ASP.NET

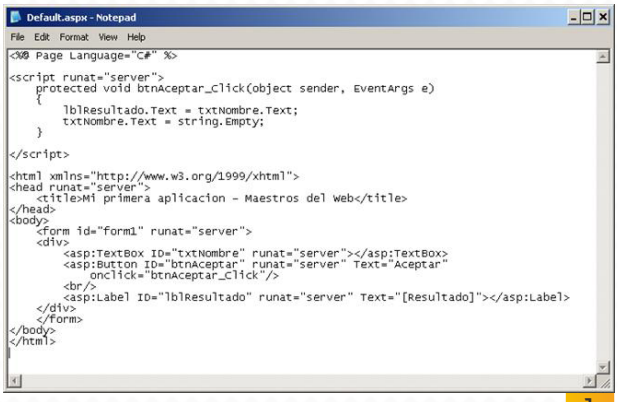
C# Código fuente

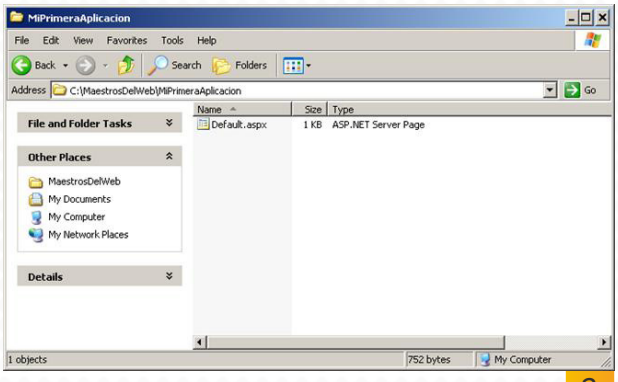
```
<%--Directiva--%>
<%@ Page Language="C#" %>
<%--Codigo en linea--%>
<script runat="server">
  protected void btnAceptar_Click(object sender, EventArgs e){
    lblResultado.Text = txtNombre.Text;
    txtNombre.Text = string.Empty;
  }
</script>
<%--HTML para dibujar los controles en pantalla--%>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server"><title>Mi primera aplicacion - Maestros del Web</
title></head>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:TextBox ID="txtNombre" runat="server"></asp:TextBox>
        <asp:Button ID="btnAceptar" runat="server" Text="Aceptar"
        onclick="btnAceptar_Click"/> <br/>
        <asp:Label ID="lblResultado" runat="server" Text="[Resultado]"></
asp:Label
      </div>
    </form>
  </body>
```

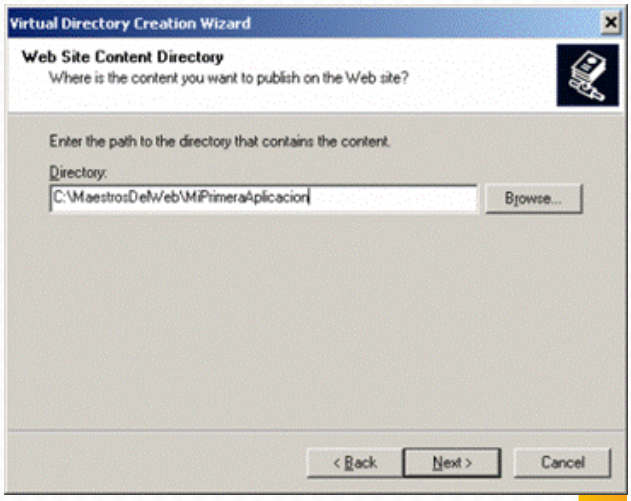
```
</html>
```

¿Cómo probamos nuestra primera aplicación?

1. Abrimos el Bloc de Notas y copiamos el código de arriba de la siguiente forma:
2. Guardamos el archivo con el nombre Default.aspx.
3. Creamos nuestro espacio virtual en el IIS
   a. Abrimos el IIS y creamos un nuevo directorio virtual.
   b. Ingresamos el nombre que tendrá nuestra aplicación web.
   c. Indicamos el directorio donde se guarda nuestra página web Default.aspx.
   d. Seguimos las instrucciones para crear nuestro directorio virtual. Al finalizar, tenemos nuestra primera aplicación ASP.NET instalada para probar. Ahora abrimos el navegador y escribimos en la barra de dirección: http://localhost/MiPrimeraAplicacionWeb/Default.aspx



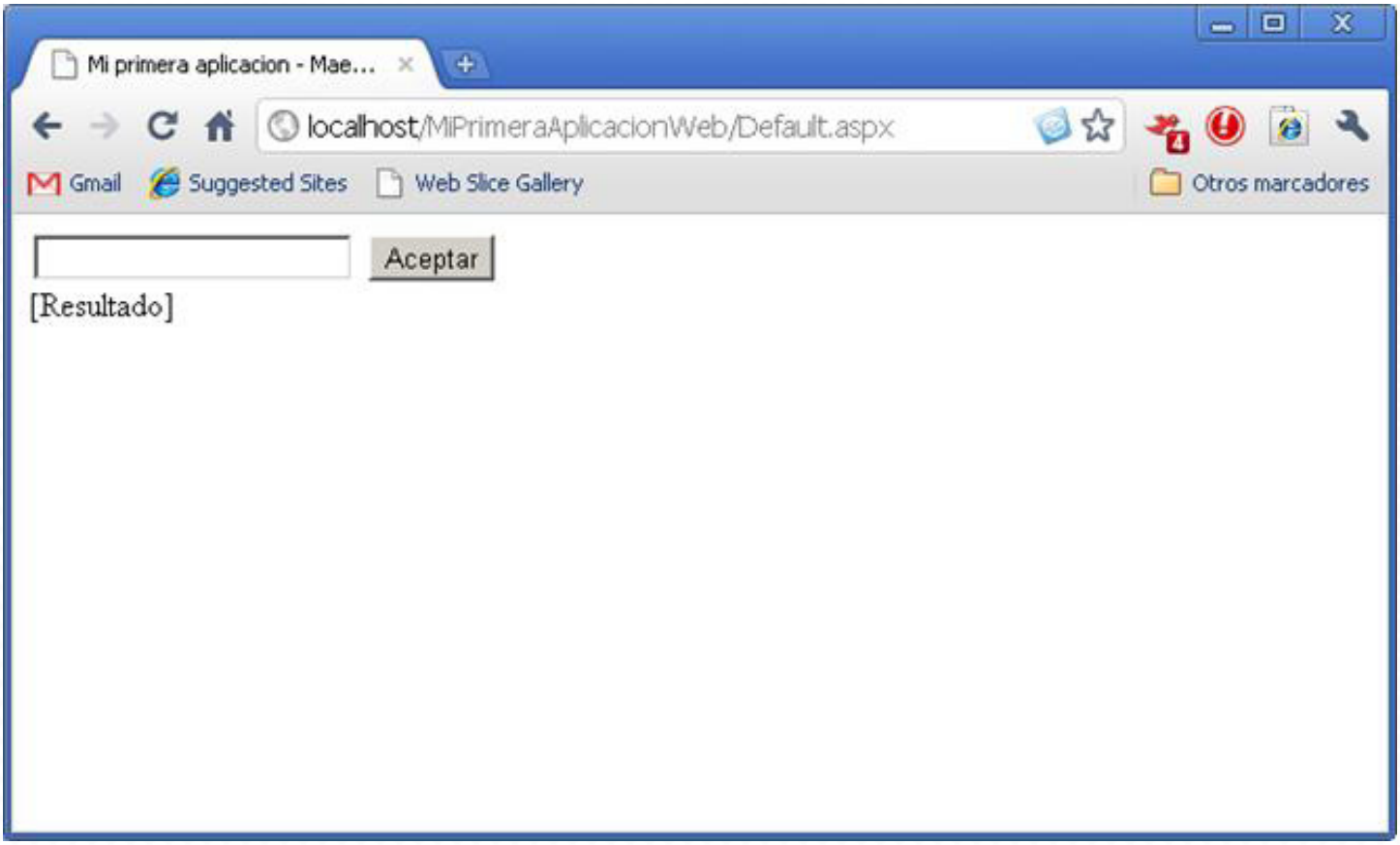




C

En pantalla vemos tres objetos

1. Un TextBox llamado txtNombre (para ingresar un nombre).
2. Un botón llamado btnAceptar con el evento OnClick definido (ejecutamos la acción de pegar contenido).
3. Un label de resultados llamado lblResultado (mostramos los resultados luego de ejecutar el evento del botón).



Estructura de nuestra aplicación web

La estructura de nuestra primera aplicación es muy simple.

1. Contiene una directiva:

```
<%@ Page Language="C#" %>
```

Le estamos indicando que la página usará lenguaje C# (C Sharp).

2. Código en línea:

C# Código fuente

```
<script runat="server">
  protected void btnAceptar_Click(object sender, EventArgs e)
  {
    lblResultado.Text = txtNombre.Text;
    txtNombre.Text = string.Empty;
  }
</script>
```

El tag script está indicando que el código se ejecutará del lado del servidor. En esta porción del código van las acciones que ejecutaremos en nuestra aplicación, en este caso sólo tenemos una acción asociada al botón "aceptar".

3. Código HTML para la creación de objetos en pantalla.

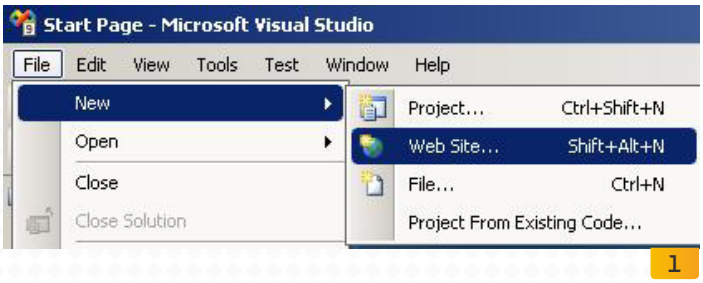
C# Código fuente

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"><title>Mi primera aplicacion - Maestros del Web</title></
head>
<body>
 <form id="form1" runat="server">
  <div>
   <asp:TextBox ID="txtNombre" runat="server"></asp:TextBox>
   <asp:Button ID="btnAceptar" runat="server" Text="Aceptar"
onclick="btnAceptar_Click"/>
   <br/><asp:Label ID="lblResultado" runat="server" Text="[Resultado]"></
asp:Label>
```

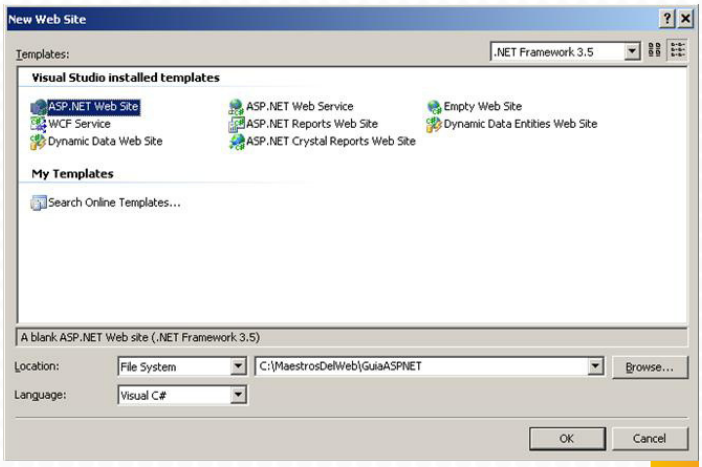
```
    </div>
  </form>
</body></html>
```

Creación de un proyecto ASP.NET con Visual Studio

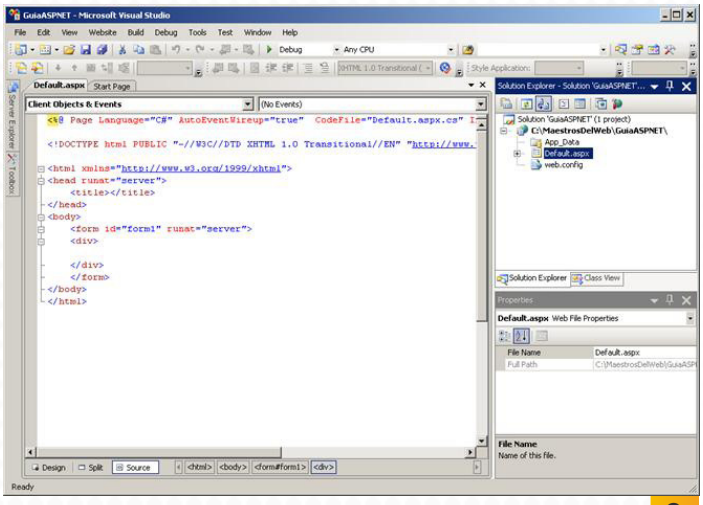
1. Abrimos Visual Studio .NET y creamos un nuevo proyecto web.
2. Seleccionamos el tipo de proyecto que deseamos realizar.
   a. Seleccionamos ASP.NET Web Site.
   b. Indicamos la ruta donde vamos a guardar el proyecto: c:\MaestrosDelWeb\GuiaASPNET
   c. Determinamos el lenguaje que vamos a utilizar: Visual C#.
   d. Por último presionamos el botón OK para crear la solución.



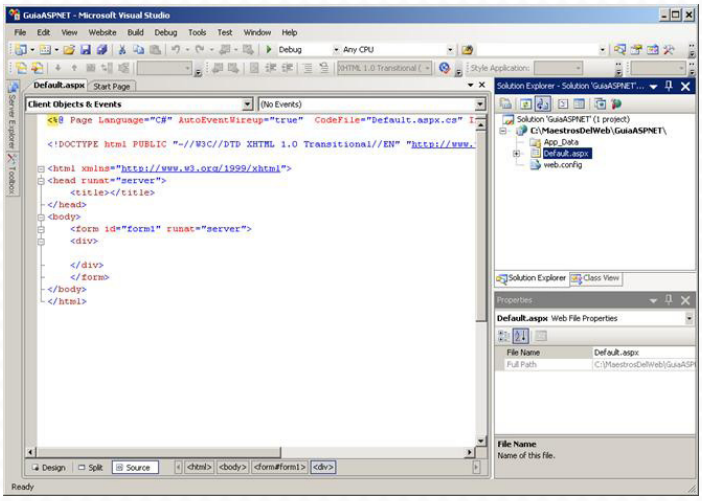
1



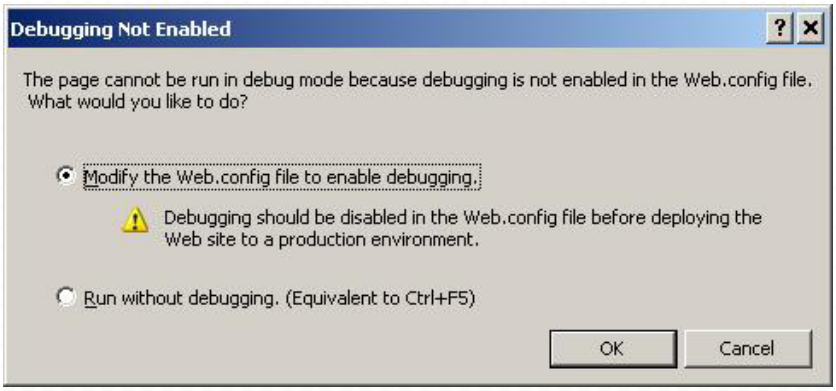
2



3



3. Una vez creada la solución el IDE genera el primer template.



En la pantalla de edición de código copiamos y pegamos el código anterior, ahora tenemos nuestra primera página web dentro del entorno de desarrollo integrado.

Para ejecutar la aplicación dentro del entorno de desarrollo presionamos la tecla F5 y nos pregunta si deseamos habilitar el modo de “debug”. Presionamos OK.

A continuación se abre nuestro navegador predeterminado ejecutando la aplicación en modo “debug”. Al trabajar con un entorno de desarrollo no es necesario crear un espacio virtual en el IIS ya que la misma aplicación se encarga de preparar el ambiente.

Capítulo 3

Ejecutar código JavaScript en ASP.NET

Capítulo 3

Ejecutar código JavaScript en ASP.NET

Luego de crear la primera aplicación web ahora vamos a ejecutar código JavaScript en ASP.NET agregando funcionalidad JavaScript a la ya codificada en C#. Una forma sencilla es agregando un atributo al control del servidor que deseamos ejecute una función JavaScript.

Dentro del TAG HEAD escribimos lo siguiente:

C# Código fuente

```
<%--Codigo JavaScript--%>
<script language="javascript" type="text/javascript">
 function fnAceptar() {
  alert('El Contenido del TextBox es: ' + document.getElementById("txtNombre").
value);
  document.getElementById("txtNombre").value = '';
 }
</script>
```

Ahora necesitamos indicarle a ASP.NET que necesitamos ejecutar esa función desde el botón.

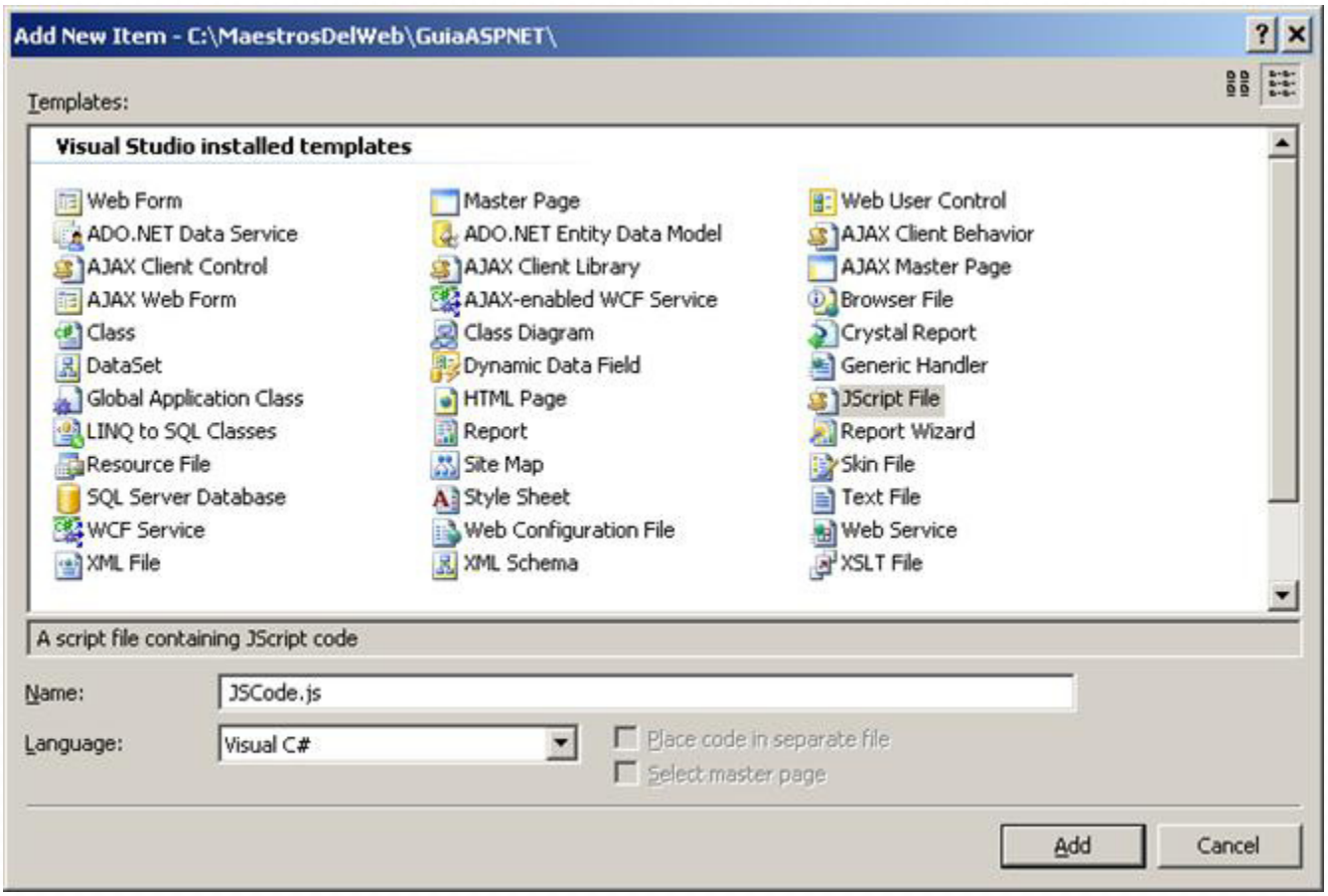
C# Código fuente

```
<script runat="server">
 protected void Page_Load(object sender, EventArgs e)
 {
  this.btnAceptar.Attributes.Add("OnClick", "javascript:return fnAceptar();");
 }
</script>
```

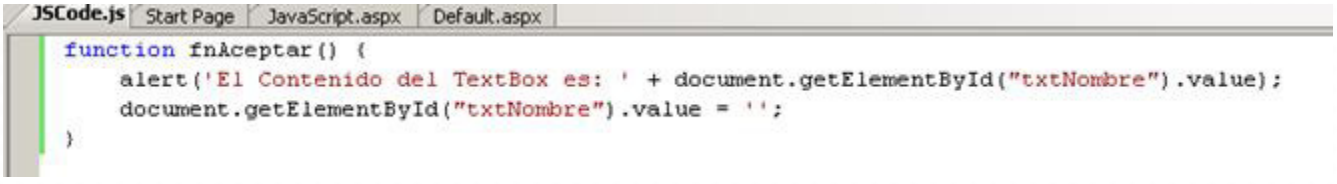
Asignamos la función fnAceptar al evento OnClick del botón. El código completo debe quedar de la siguiente forma:

```
<%--Directiva--%>
<%@ Page Language="C#" %>
<%--Codigo en linea--%>
<script runat="server">
 protected void Page_Load(object sender, EventArgs e)
 {
  this.btnAceptar.Attributes.Add("OnClick", "javascript:return fnAceptar();");
 }
</script>
<%--HTML para dibujar los controles en pantalla--%>
<html xmlns="http://www.w3.org/1999/xhtml">
 <head id="Head1" runat="server">
  <title>Mi primera aplicacion - Maestros del Web</title>
  <%--Codigo JavaScript--%>
  <script language="javascript" type="text/javascript">
  function fnAceptar() {
   alert('El Contenido del TextBox es: ' + document.
getElementById("txtNombre").value);
   document.getElementById("txtNombre").value = '';
  }
  </script>
 </head>
<body>
<form id="form1" runat="server">
 <div>
  <asp:TextBox ID="txtNombre" runat="server"></asp:TextBox>
  <asp:Button ID="btnAceptar" runat="server" Text="Aceptar"/>
  <br/>
  <asp:Label ID="lblResultado" runat="server" ></asp:Label>
 </div>
</form>
</body>
</html>
```

Revisemos la manera en la que conviven los códigos C# y JavaScript en un mismo archivo .aspx . Ahora, separemos el código en distintos archivos para mejorar el mantenimiento y lectura, agregando un archivo JavaScript a nuestro proyecto ASPNET, necesitamos mover el código JavaScript al archivo y referenciarlo desde nuestra página .aspx.



Llamaremos a nuestro archivo JSCode.js abrimos el archivo y copiamos el código JavaScript que inicialmente escribimos en nuestra página .aspx.



Ahora eliminamos el código JavaScript de la página .aspx y hacemos referencia al archivo .js dentro del tag HEAD de esta forma evitamos llenar nuestro .aspx con código que hace difícil la lectura. También, se puede separar el código C# de una forma similar al que acabamos de ver con JavaScript, pero lo vere-

mos en unos capítulos más adelante.

ClientScript.RegisterStartupScript

Una forma de usar código JavaScript desde una página ASP.NET es utilizando el método *RegisterStartupScript* de la clase *ClientScript*, que nos permite registrar un script en tiempo de ejecución para su posterior utilización.

Ejemplo: Creamos la función en C#

C# Código fuente

```
private void RegistrarScript()
{
 const string ScriptKey = "ScriptKey";
 if (!ClientScript.IsStartupScriptRegistered(this.GetType(), ScriptKey))
 {
  StringBuilder fn = new StringBuilder();
  fn.Append("function fnAceptar() { ");
  fn.Append("alert('El Contenido del TextBox es: ' + document.getElementById
  fn.Append("document.getElementById(\"txtNombre\").value = '';");
  fn.Append("}");
  ClientScript.RegisterStartupScript(this.GetType(),
  ScriptKey, fn.ToString(), true);
  }
 }
```

Es la misma función usada anteriormente *fnAceptar()* ahora registrándola desde ASP.NET. Luego hacemos la llamada a la función desde el *Page_Load()*.

```
RegistrarScript();
```

Sin necesidad de un archivo *.js* ni escribir el código JavaScript en la sección de HTML. Código completo:

C# Código fuente

```
<%--Directiva--%>
<%@ Page Language="C#" %>
<%--Codigo en linea--%>
<script runat="server">
 protected void Page_Load(object sender, EventArgs e)
 {
  if (!IsPostBack)
  {
   this.btnAceptar.Attributes.Add("OnClick", "javascript:return fnAceptar();"
   RegistrarScript();
  }
 }
 private void RegistrarScript()
 {
  const string ScriptKey = "ScriptKey";
  if (!ClientScript.IsStartupScriptRegistered(this.GetType(), ScriptKey))
  {
   StringBuilder fn = new StringBuilder();
   fn.Append("function fnAceptar() { ");
   fn.Append("alert('El Contenido del TextBox es: ' + document.
getElementById(\"txtNombre\"
   fn.Append("document.getElementById(\"txtNombre\").value = '';");
   fn.Append("}");
   ClientScript.RegisterStartupScript(this.GetType(),
   ScriptKey, fn.ToString(), true);
  }
 }
</script>
```

```
<%--HTML para dibujar los controles en pantalla--%>
<html xmlns="http://www.w3.org/1999/xhtml">
 <head id="Head1" runat="server">
  <title>Mi primera aplicacion - Maestros del Web</title>
 </head>
<body>
 <form id="form1" runat="server">
  <div>
   <asp:TextBox ID="txtNombre" runat="server"></asp:TextBox>
   <asp:Button ID="btnAceptar" runat="server" Text="Aceptar"/>
   <br/>
   <asp:Label ID="lblResultado" runat="server" ></asp:Label>
  </div>
 </form>
</body>
</html>
```

Capítulo 4

Archivos Global.asax y Web.Config

Capítulo 4

Archivos Global.asax y Web.Config

El archivo Global.asax es opcional y nos permite manejar eventos de aplicación y de Session en nuestra aplicación, el archivo Global.asax reside en el directorio raíz de una aplicación de ASP.NET. Este archivo se compila en tiempo de ejecución, no es accesible por URL y debe ubicarse en el directorio raíz de la solución.

Template Global.asax:

C# Código fuente

```
<%@ Application Language="C#" %> <script runat="server">
void Application_Start(object sender, EventArgs e) {
// Codigo que se ejectura cuando inicia la aplicacion
}
void Application_End(object sender, EventArgs e) {
// Codigo que se ejectura cuando finaliza la aplicacion
}
void Application_Error(object sender, EventArgs e) {
// Codigo que se ejectura cuando ocurre algun error en la aplicacion
}
void Session_Start(object sender, EventArgs e) {
// Codigo que se ejectura cuando se inicia la sesion.
}
void Session_End(object sender, EventArgs e) {
// Codigo que se ejectura cuando finaliza la sesion.
}
</script>
```

El uso clásico es el contador de visitas que debe sumar por cada usuario que ingrese al sitio web, el evento que se ejecuta por cada ingreso es Session_Start.

Ejemplo: código en Gobal.asax

Inicializamos una variable en 0 (cero) al iniciar la aplicación y luego por cada inicio de sesión la incrementamos y la cargamos a una variable de sesión para acceder a ella desde el resto de la aplicación.

C# Código fuente

```
private int cnt; void Application_Start(object sender, EventArgs e) { // Codigo
que se cnt = 0;
} void Session_Start(object sender, EventArgs e) { // Codigo que se ejectura
cuando se Session["cv"] = cnt++;
} Código de la pagina aspx a la que llamaremos ContadorVisitas.aspx <%@ Page
Langua
protected void Page_Load(object sender, EventArgs e) {
Response.Write(Session["cv"]); </script>
}
<html xmlns="http://www.w3.org/1999/xhtml"> <head runat="server">
<title></title> </head>
<body> <form id="form1" runat="server"> <div>
</div>
</form> </body>
</html>
```

La ventaja de manejarlo de esta forma es que vamos a lograr un contador de visitas reales, es decir, sólo sumará cada inicio de sesión en lugar de sumar cada solicitud a la página. Para probarlo copiamos el archivo *ContadorVisitas.aspx* y *Global.asax* al directorio virtual creado anteriormente en el IIS y lo ejecutamos:

`http://localhost/MiPrimeraAplicacionWeb/ContadorVisitas.aspx.`

Web.config

El archivo Web.config es el archivo principal de configuraciones de toda aplicación ASP.NET, es un archivo XML que controla el funcionamiento del sitio web.

- Control de seguridad.
- Conexiones a bases de datos.
- Estado de las sesiones.
- Control de errores.
- Configuraciones personalizadas.
- Al igual que el global.asax debe ser colocado en la raíz de la aplicación.

Estructura básica de un archivo *web.config*:

```
<configuration> <system.web/>
<appSettings/> </configuration>
```

Agregamos Keys a nuestro *web.config* para ver cómo usarlos en nuestro sitio.

Por ejemplo, si queremos controlar que el contador de visitas este activo o no, agregamos una key llamada *HabilitarContadorVisitas* y lo seteamos como verdadero. Nuestras Keys (Claves) se agregan en el tag *<appSettings>*

```
<appSettings> <add key="HabilitarContadorVisitas" value="true" /> </
appSettings>
```

Código del evento Session_Start

C# Código fuente

```
void Session_Start(object sender, EventArgs e) { // Codigo que se ejectura
cuando se inicia if (Convert.ToBoolean(ConfigurationManager.AppSettings.
Get("HabilitarContadorVisitas"
 Session["cv"] = cnt++;
}
```

Luego copiamos al directorio virtual los archivos modificados y veremos que el contador de visitas siempre nos marcara 0 (cero). Más adelante usaremos el web.config para establecer nuestra conexión con datos.

Utilización de código detrás del modelo o código en línea

Los ejemplos que hemos visto hasta el momento son utilizando código en línea (Code inline). Usar código en línea es muy práctico ya que nos permite tener los controles del formulario y el código de los eventos en un mismo archivo. La desventaja es que en la mayoría de los proyectos se trabaja con más de un desarrollador y diseñador gráfico. Para esto es más útil manejar la interfaz de usuario (UI) y la programación por separado.

Código detrás del modelo (Code Behind)

Manejar código detrás del modelo nos permite organizar los eventos en forma separada, todo lo relacionado con interfaz de usuario lo manejamos en el archivo .aspx y el control de los eventos en un archivo separado .cs (para C Sharp). De forma similar a la que manejamos los archivos de JavaScript (.js) donde incluimos todas las funciones y luego las referenciamos en el aspx hacemos con el Code Behind.

Recordemos la primer directiva de nuestra página aspx.

```
<%@ Page Language="C#" %>
```

Indicamos que el código de eventos va separado de la UI y lo marcamos de la siguiente manera:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default2.aspx.cs"
Inherits="Default2" %>
```

¿Qué le estamos diciendo a ASP.NET al incluir esta directive Page a la página aspx?

Lo vemos punto por punto:

- Language="C#": este atributo le está indicando al compilador que el código estará escrito en lenguaje C Sharp.
- AutoEventWireup="true" : indica si el código será inline o Behind. Al setearlo en "true" le estamos indicando que el código será detrás del modelo (CodeBehind) y se especifica el nombre del archivo donde buscará el código de eventos.

- CodeFile="Default2.aspx.cs": indica el nombre del archivo donde incluiremos el código de eventos.
- Inherits="Default2": se especifica la clase a heredar, esta clase la busca dentro del archivo que indicamos en CodeFile.

Ejemplo de código utilizando Code Behind

Nuestro ejemplo tendrá dos archivos:

1. *Default.aspx*: incluirá todo lo relacionado con interfaz de usuario.
2. *Default.cs*: código de los eventos que se ejecuten desde el formulario.

Default.aspx

C# Código fuente

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits=
<html> <head runat="server">
<title></title> </head>
<body> <form id="form1" runat="server">
 <div> <asp:TextBox ID="txtNombre" runat="server"></asp:TextBox> <asp:Button
ID="btnAceptar"
onclick="btnAceptar_Click"/> <br/>
 <asp:Label ID="lblResultado" runat="server" Text="[Resultado]"></asp:Label> </
div>
</form> </body>
</html>
```

Default.cs

C# Código fuente

```
using System; using System.Web.UI.WebControls;
public partial class Default : System.Web.UI.Page {}
protected void btnAceptar_Click(object sender, EventArgs e) {}
lblResultado.Text = txtNombre.Text; txtNombre.Text = string.Empty;
```

Capítulo 5

Controles de servidor y eventos

Capítulo 5

Controles de servidor y eventos

Los controles de servidor son identificados con la etiqueta runat=”server” y por medio de su ID podemos programar en tiempo de ejecución mediante código C Sharp. Muchos de los controles de servidor son similares a controles HTML como los cuadros de texto o botones, pero hay controles de servidor muchos más complejos que brindan numerosas funcionalidades: grillas, calendarios, vista de árbol, menú y hasta controles de conectividad.

¿Qué son los eventos?

Básicamente un evento responde a una acción que sucede en relación al objeto, los que suceden en la interfaz de usuario son los más usuales. Por ejemplo: el evento OnClick se produce al presionar el botón y se ejecuta el código que ingresamos para ese evento.

Agregar controles de servidor en tiempos de diseño:

Incluir controles a una página web ASP.NET se hace de la misma forma que los elementos HTML. Puede utilizar un diseñador visual y agregar un control del cuadro de herramientas, o bien, puede escribir el elemento que representa el control.

```
<asp:Button ID=”btnAceptar” runat=”server” Text=”Aceptar”
onclick=”btnAceptar_Click”/>
```

Los controles de servidor web se declaran con una etiqueta XML que hace referencia al espacio de nombres asp. El inicio y fin del tag está determinado por <asp />

- runat=”server”: nos indica que se trata de un control de servidor.
- onclick=”btnAceptar_Click”: contiene el código que se va a ejecutar cuando se produzca el evento onclick.

Agregar controles de servidor en tiempos de ejecución:

Muchas veces es necesario crear un control en tiempo de ejecución en lugar de hacerlo en tiempo de diseño. Supongamos que dinámicamente debemos crear varios combos pero no sabemos la cantidad hasta que la página vaya recibiendo información. En este caso es necesario crearlo mediante código indicando la cantidad de veces que necesitamos crearlo.

Ejemplo de creación de un TextBox por programación.

```
TextBox txt = new TextBox();
txt.ID = "txtNombre";
txt.Text = "MDW";
```

Ahora este mismo TextBox lo vamos a incluir en un Panel.

```
Panel pnl = new Panel();
pnl.ID = "Panel";
pnl.Controls.Add(txt);
```

Cuando ejecutemos nuestra página veremos que en pantalla se crea un cuadro de texto con el contenido "MDW" dentro de un control PANEL. En la siguiente rutina veremos cómo agregar dinámicamente controles de servidor en tiempo de ejecución.

Creamos un archivo llamado Controles.aspx y copiamos el siguiente código:

C# Código fuente

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Controles.aspx.cs"
<html xmlns="http://www.w3.org/1999/xhtml"> <head runat="server">
<title></title> </head>
<body> <form id="form1" runat="server"> <div>
<asp:Label ID="lblCantidad" runat="server" Text="Ingrese un numero"></
asp:Label> <asp:TextBox onclick="btnProceso_Click" /> </div>
</form> </body></html>
```

Ahora creamos otro archivo llamado Controles.aspx.cs y copiamos el siguiente código:

C# Código fuente

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class Controles : System.Web.UI.Page {}
protected void Page_Load(object sender, EventArgs e) {}
protected void btnProceso_Click(object sender, EventArgs e) {
Table tbl = new Table(); try
{
int cantidad = Convert.ToInt32(txtCantidad.Text.Trim());
Label lbl; for (int i = 0; i < cantidad; i++) {
lbl = new Label(); lbl.ID = "IdLabel" + i.ToString(); lbl.Text = "Label Nro: "
AgregarLabel(tbl, lbl); Page.Controls.Add(tbl);
}
} catch (Exception ex) {
Label lbl = new Label(); lbl.ID = "error"; lbl.Text = ex.Message;
AgregarLabel(tbl, lbl); }
private void AgregarLabel(Table t,Label l) {}
}
TableCell tc = new TableCell(); TableRow tr = new TableRow();
tc.Controls.Add(l); tr.Cells.Add(tc); t.Rows.Add(tr);
```

Acabamos de ver como generamos controles de servidor por programación, ahora vamos a ver como trabajamos con eventos en forma dinámica.

C# Código fuente

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Controles.aspx.cs"
<html xmlns="http://www.w3.org/1999/xhtml"> <head runat="server">
<title></title> </head>
<body> <form id="form1" runat="server"> <div>
<asp:Label ID="lblCantidad" runat="server" Text="Ingrese un numero"></
```

```
asp:Label> <asp:TextBox </div>
</form> </body> </html>
```

Tenemos el mismo formulario a diferencia de que el botón no tiene el evento OnClick. Ahora lo vamos a hacer en tiempo de ejecución. Esto es equivalente a onclick=”btnProceso_Click” pero por programación .btnProceso. Click += new EventHandler(btnProceso_Click);

Este código se agrega en el evento Page_Load que se ejecuta cada vez que se carga la página.

```
protected void Page_Load(object sender, EventArgs e) { if(!IsPostBack)
btnProceso.Click += new EventHandler(btnProceso_Click);}
```

Se agrega un concepto nuevo en el evento Page_Load “IsPostBack” se usa para saber si ya se ha ejecutado el método Page_Load. Como en este caso sólo necesitamos que se declare el sólo una vez preguntamos si es la primera vez que se ejecuta mediante if(!IsPostBack).

Las veces posteriores que se cargue la página no entrarán en ese IF. El signo ! es el negado de la función, es como preguntar “¿no es postback?“.

Controles de usuarios reutilizables

ASP.NET provee controladores Web y HTML pero también puedes crear los propios, los controladores de usuario tiene la extensión .ascx cuya sintaxis declarativa es similar a la que se emplea para crear páginas web. Hasta el momento usábamos la declarativa @Page ahora necesitamos usar @Control para identificarlo como control de usuario personalizado.

Algo muy interesante de los controles de usuarios además de poder agregar toda la funcionalidad necesaria, podemos usarlo en cualquier WebForms del sitio y al estar centralizados cualquier modificación que realicemos se verá reflejado en todo el sitio donde estemos usando el @Control.

Ejemplo en código de un control de usuario:

Vamos a llamar a nuestro control *UCTextBox.ascx*

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="UCTextBox.ascx.cs"
%>
```

Con la directiva @ Control ASP.Net ya entiende que se trata de un control de usuario y su correspondiente CodeBehind llamado UCTextBox.ascx.cs.

Hasta el momento sólo estamos indicando a ASP.NET que es un control de usuario, aún no le indicamos lo que hará. Agregamos un control de servidor TextBox y lo vamos a personalizar.

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

Para que el ejemplo sea sencillo le daremos una funcionalidad nueva que el TextBox que nos provee ASP.NET no tiene. Determinar si el contenido del TextBox es numérico y hacer algo al respecto.

Agregaremos algunas propiedades al TextBox:

```
<asp:TextBox ID="TextBox1" runat="server" AutoPostBack="True"
ontextchanged="TextBox1_ TextChanged"></asp:TextBox>
```

Propiedades necesarias para el ejemplo:

- ontextchanged="TextBox1_TextChanged" : como vimos en el capítulo de eventos, vamos a ejecutar ese método cuando cambie el contenido del textbox.
- AutoPostBack="True" : al estar seteado en True hace que se realice un postback y vaya al servidor.

Contenido del UCTextBox.ascx.cs

C# Código fuente

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
```

```
using System.Text;
public partial class UCTextBox : System.Web.UI.UserControl {
protected void Page_Load(object sender, EventArgs e) {
}
//Atributo
private bool _esnumerico = false; //Propiedad public bool EsNumerico {
get { return this._esnumerico; } set { this._esnumerico = value; }
}
protected void TextBox1_TextChanged(object sender, EventArgs e) {
if (this.EsNumerico && !IsNumeric(TextBox1.Text.Trim())) {
this.TextBox1.Text = string.Empty; return;
}
//Metodo publico que evalua si el valor ingresado es numero o no.
public bool IsNumeric(object Expression) {
bool isNum; double retNum;
isNum = Double.TryParse(Convert.ToString(Expression), System.Globalization.
NumberSty
}
}
}
return isNum;
```

¿Cómo funciona?

Todo está atado a la propiedad pública bool EsNumerico en caso que este seteada en TRUE el control va a responder de determinada manera. El EventHandler TextBox1_TextChanged pregunta si EsNumerico es verdadero, además el contenido del textbox no es número entonces limpia el control y no continua con la acción.

```
protected void TextBox1_TextChanged(object sender, EventArgs e) {
if (this.EsNumerico && !IsNumeric(TextBox1.Text.Trim())) {
this.TextBox1.Text = string.Empty; return;
}}
```

Incertaremos el Control de usuario en un WebForm de la misma forma que lo hacemos con un Control

web lo hacemos con un Control de usuario pero con algunas variantes.Lo primero que debemos hacer es declarar el Control de usuario en el ASPX.

```
<%@ Register src="UCTextBox.ascx" tagname="UCTextBox" tagprefix="uc1" %>
```

El tagprefix es importante ya que ese tag reemplazará al .asp que tienen los controles web que nos brinda ASP.NET.

```
<uc1:UCTextBox ID="UCTextBox1" runat="server" />
```

Ahora en nuestra página web veremos implementado el Control de usuario.

¿Qué más nos falta? debemos determinar si la propiedad publica EsNumerico la dejamos en False que es como la seteamos por default o en True para que tome acciones sobre el contenido del textbox.

Agregamos esta línea en el Page_Load del webform.

```
this.UCTextBox1.EsNumerico = true;
```

Muchas veces necesitamos cargar un control de usuario dinámicamente y en tiempo de ejecución, para ellos usaremos la siguiente sintáxis.

```
Control MiControl = LoadControl("UCTextBox.ascx");
```

Crear un página de login, autenticación y seguridad

ASP.NET nos brinda controles de servidor complejos que nos facilitan la implementación en nuestras aplicaciones. Un buen ejemplo es el control de Login.

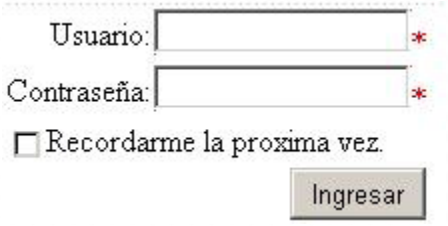
Revisemos las propiedades principales que ofrece para adaptarlo a nuestras necesidades.

```
<asp:Login ID="Login1" runat="server"
LoginButtonText="Ingresar"
PasswordLabelText="Contraseña:" RememberMeText="Recordarme la proxima vez."
TitleText=
UserNameLabelText="Usuario:">
</asp:Login>
```

Propiedades que utilizamos son:

- *LoginButtonText* : nos permite modificar el texto del botón.
- *PasswordLabelText* : texto de la solicitud de contraseña.
- *UserNameLabelText* : texto de la solicitud de nombre de usuario.
- *RememberMeText* : texto de recordatorio.

El control nos queda de esta manera:



Ahora que cambiamos el idioma usando las propiedades del control, darle funcionalidad a través de sus

eventos. El Evento principal es Authenticate donde se evalua si el usuario y contraseña son correctos de la siguiente forma:

Agregamos el *EventHandler onauthenticate="Login1_Authenticate"* a nuestro control Login.

Debe quedar de esta forma en el webform:

```
<asp:Login ID="Login1" runat="server" onauthenticate="Login1_Authenticate"
LoginButtonText= "Ingresar" PasswordLabelText="Contraseña:"
RememberMeText="Recordarme la proxima vez." TitleText=""
UserNameLabelText="Usuario:"> </asp:Login>
```

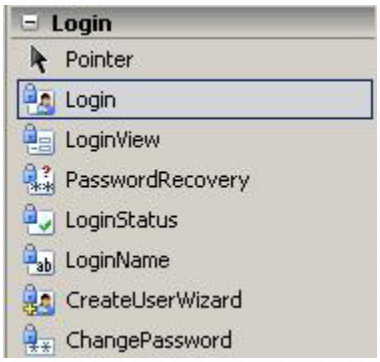
Ahora veremos el código que se ejecutará al querer autenticarse.

C# Código fuente

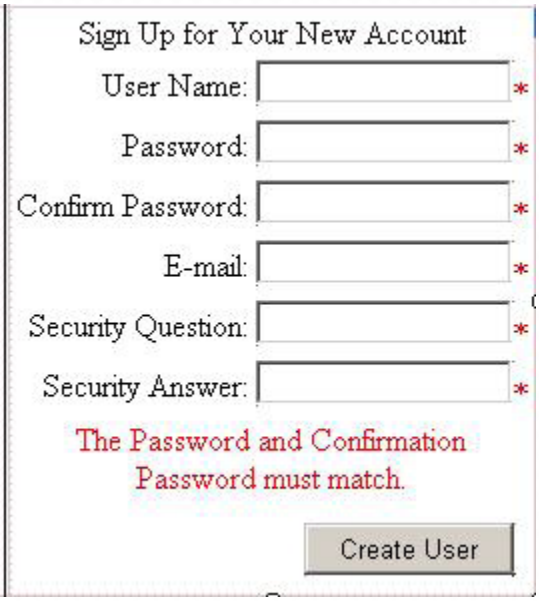
```
protected void Login1_Authenticate(object sender, AuthenticateEventArgs e) {
bool Autenticado = false;
Autenticado = LoginCorrecto(Login1.UserName, Login1.Password);
e.Authenticated = Autenticado; if (Autenticado) {
Response.Redirect("Default.aspx");
}
private bool LoginCorrecto(string Usuario, string Contrasena) { if (Usuario.
Equals(
return true; return false;
}
```

En este ejemplo estoy prefijando en el código un usuario y password ("maestros","delweb"). En estos casos el método privado LoginCorrecto por lo general va a una base de datos donde están almacenados los usuarios y passwords y recién se verifica si es correcto.

Otros controles relacionados a login son:



CreateUserWizard que nos da todas las facilidades para dar de alta nuevos usuarios.



Capítulo 6

Utilización de estilos en ASP.NET (CSS)

Capítulo 6

Utilización de estilos en ASP.NET (CSS)

ASP.NET nos permite factorizar la información de estilo y diseño en un grupo separado de ficheros, un Tema (Theme) se puede aplicar a cualquier sitio de forma que afecte a la apariencia y entorno de las páginas y controles del sitio. Los cambios en el Estilo (CSS) de un sitio pueden administrarse realizando cambios al Tema sin editar las páginas de forma individual.

Los Temas también se pueden compartir con otros desarrolladores. Hay varias formas de aplicar Estilos a una página web en ASP.NET pero no todas son las más óptimas. Todo control de servidor tiene atributos que nos permiten cambiar su aspecto.

```
<asp:Label ID="Label1" runat="server"
BackColor="#FF6600"
BorderColor="#00CC00"
BorderStyle="Solid"
Font-Bold="True"
Font-Names="Arial"
Font-Size="10pt"
ForeColor="White"
Text="Mi label"></asp:Label>
```

Aplicando el diseño nuestro Label se ve de esta forma:

Aplicar estilos mediante atributos tiene sus ventajas y desventajas, por un lado la comodidad y velocidad de aplicar un estilo particular a un control determinado. Pero por el otro, al no estar centralizado, no te permite reutilizar el estilo para aplicar a otros controles y si has aplicado este estilo a varios controles y deseas cambiarlo deberás hacerlo uno por uno.

La mejor opción es utilizar Hojas de estilo “Cascading Style Sheets” (CSS) factorizar los ajustes de estilo de forma separada de las propiedades de los controles. Veamos cómo podemos aplicar el mismo estilo utilizando hojas de estilo CSS. Lo primero que vamos a hacer es crear un archivo con la extensión CSS. (“StyleSheet.css”).

En este archivo tendremos la definición del estilo, agregamos la siguiente definición a nuestro archivo .CSS:

```
.MiLabel
{
border-color: #00CC00;
border-style: solid;
background-color: #FF6600;
color: #FFFFFF;
font-family: Arial;
font-size: 10pt;
font-weight: bold;
}
```

Luego, creamos la referencia a la hoja de estilo en nuestro aspx.

```
<link href="StyleSheet.css" rel="stylesheet" type="text/css" />
```

Por último indicamos a nuestro control Label que utilizara la clase MiLabel.

```
<asp:Label ID="Label1" runat="server"
CssClass="MiLabel" Text="Mi label"></asp:Label>
```

De esta manera cada control que necesitemos aplicarle el mismo estilo simplemente deberá indicarle a la propiedad CssClass el nombre del estilo que se desea aplicar.

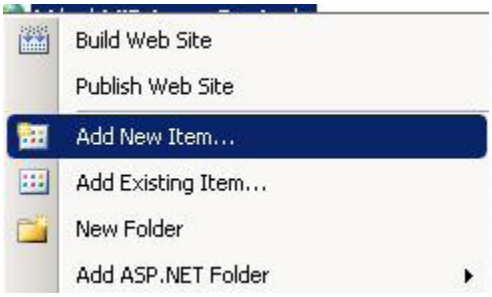
Trabajando con XML y Web Services

Los servicios web XML permiten intercambiar datos en forma de mensajes XML entre sistemas heterogéneos. Los servicios web se crean utilizando el entorno de aplicación ASP.NET, lo que permite tener acceso a numerosas características de .NET Framework.

¿Cómo crear un web services usando el IDE de ASP.NET?

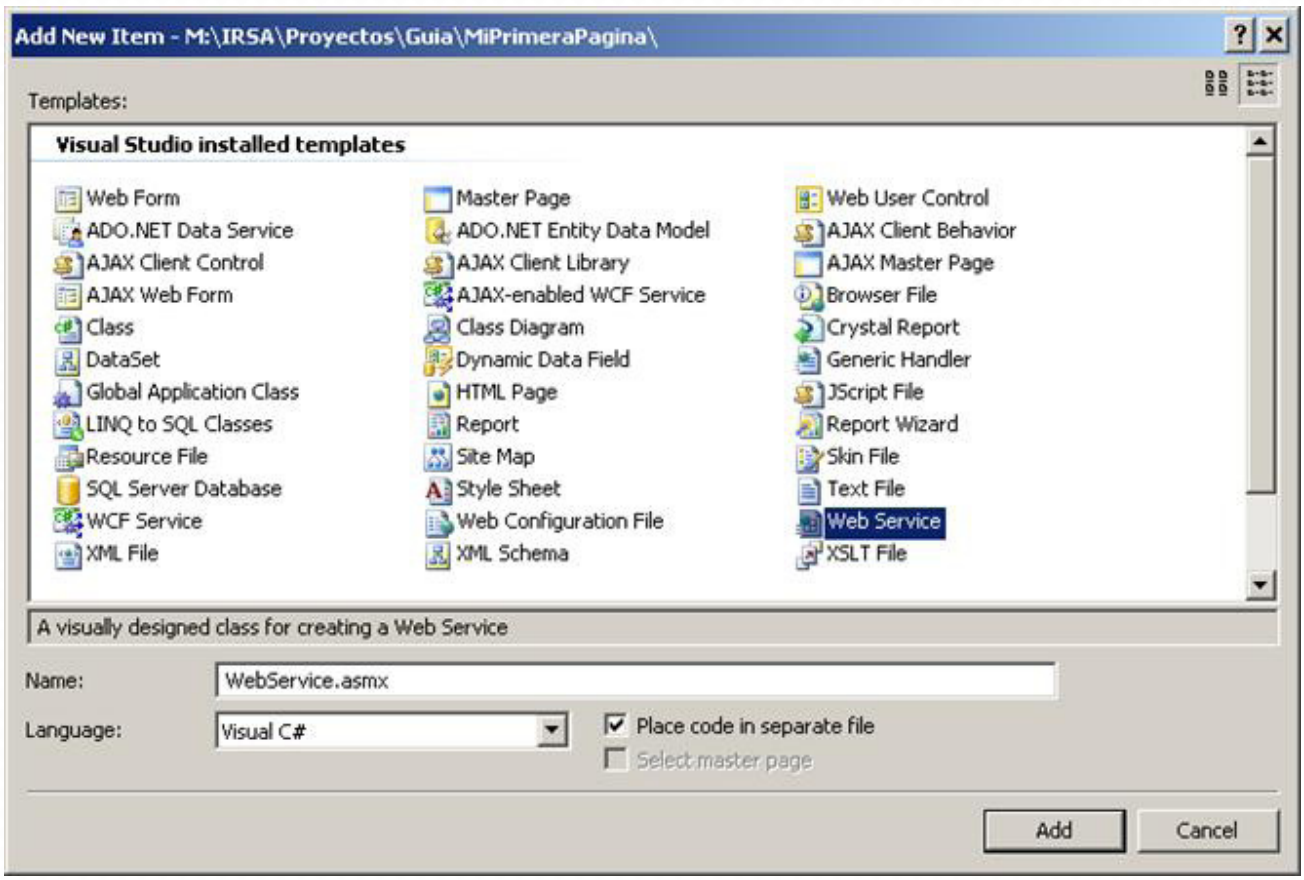
1. Lo primero que debemos hacer es agregar un nuevo ítem de tipo WebService a la solución.

Hacemos botón derecho sobre el proyecto: Add new ítem.



De la lista de ítems seleccionamos Web Service que por default nos propone un nombre. Lo vamos a cambiar para no confundir los conceptos.

Nuestro Web Services se llamará *wsMDW.asmx*



Ahora tenemos en nuestro proyecto dos archivos nuevos.

- wsMDW.asmx
- wsMDW.cs: este archivo contiene los métodos y objetos que necesitamos publicar.

wsMDW.cs

Esta es la estructura base de la clase wsMDW, esta clase hereda de System.Web.Services.WebService y de esta forma la identificamos como Web Services.

C# Código fuente

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.Services;
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class wsMDW : System.Web.Services.WebService {
 public wsMDW ()
 {} AQUI VAN LOS METODOS WEB QUE VAMOS A PUBLICAR
}
```

Nuestro primer método web:

Agregamos un método llamado "FechaYHora" que devuelve la fecha y hora del servidor.

```
[WebMethod] public string FechaYHora() {
return DateTime.Now.ToString();
}
```

Vemos que tiene la misma apariencia de un método común y corriente. Las dos grandes diferencias son que este método está dentro de una clase que hereda de System.Web.Services.WebService y tiene el Decorate [WebMethod] que lo identifica como método web. Sencillamente lo que responde un Web Services es en formato XML, procedemos a ejecutarlo presionando F5 desde el IDE de ASP.NET

Hacemos clic sobre el enlace FechaYHora veremos la respuesta que nos da el web services. El resultado es este XML con el tag string cuyo valor es la fecha y hora del momento en que se ejecuto el método web.

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">29/10/2010 11:18:26 p.m.</string>
```

De la misma forma se crean y prueba otros métodos web que reciban o no parámetros. Ahora veamos un ejemplo simple de un método web que recibe parámetros.

Probemos con el clásico Hola:

```
[WebMethod]
public string Hola(string Nombre)
{return "Hola " + Nombre; }
```

- FechaYHora
- Hola

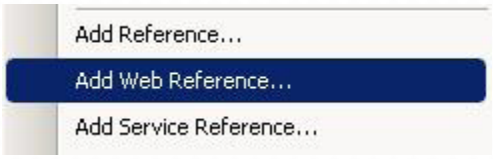
Seleccionamos el enlace Hola y nos va a pedir el parámetro Nombre que definimos en nuestro método web.

Agregamos un nombre y veamos la respuesta.

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Hola Fernando</string>
```

De esta forma testeamos web services, nos interesa ver como los podemos utilizar dentro de nuestra aplicación. Lo primero que vamos a hacer es generar una referencia web de la siguiente forma.

1. Hacemos botón derecho sobre el proyecto y agregamos una referencia web.

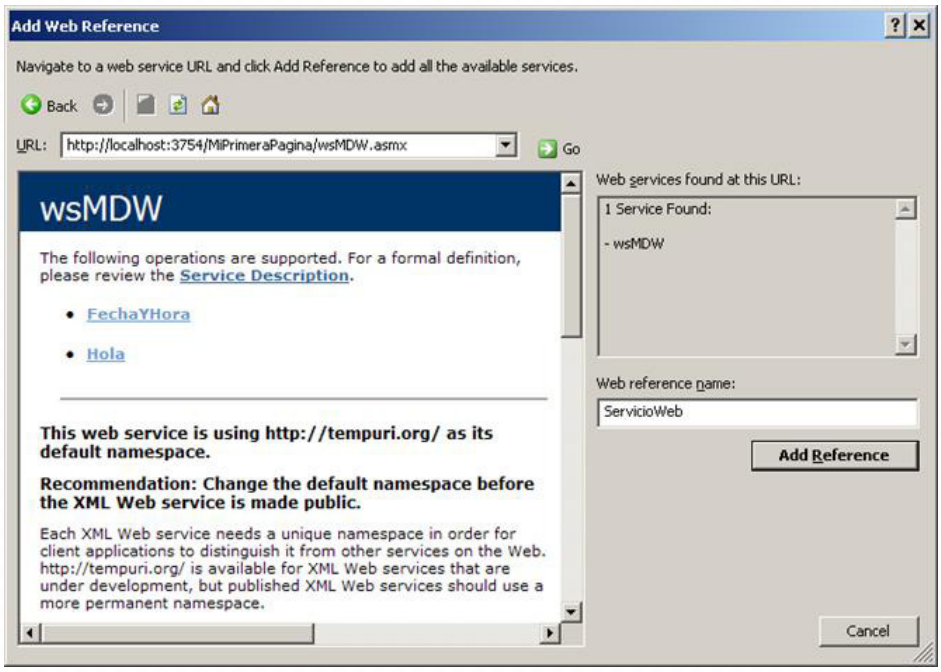


En esta ventana nos da 3 opciones.

2. Como el web services está dentro del proyecto vamos a seleccionar la primera opción.

Se listan todos los web services que tengamos en nuestro proyecto.

3. Seleccionamos el Servicio wsMDW y muestra un preview y solicita un nombre, pondremos ServicioWeb.



4. Por último, agregamos el servicio presionando el botón Add reference y se agrega la referencia a nuestro proyecto. Como vemos se creo una nueva carpeta App_WebReferences, ServicioWeb y el archivo wsMDW.discomap.

Ahora que ya tenemos referenciado nuestro web services en el proyecto podemos consumirlo. Supongamos que queremos invocar el método FechaYHora y mostrarlo en pantalla. Para eso, vamos a crear un nuevo webform llamado ConsumoWS.aspx, y agregamos un Label donde vamos a mostrar la respuesta del método.

ConsumoWS.aspx

C# Código fuente

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ConsumoWS.aspx.cs"
Inherits=
<html> <head runat="server">
<title></title> </head>
<body> <form id="form1" runat="server"> <div>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label> </div>
</form> </body>
</html>
```

Ahora la llamada al método web la hacemos desde el CodeBehind.

ConsumoWS.aspx.cs

Agregamos estas líneas al método Page_Load:

```
protected void Page_Load(object sender, EventArgs e) { ServicioWeb.wsMDW miws =
ServicioWeb.wsMDW();
Label1.Text = miws.FechaYHora();
}
```

Así de simple se crea y consume un web services realizado en ASP.NET.

Capítulo

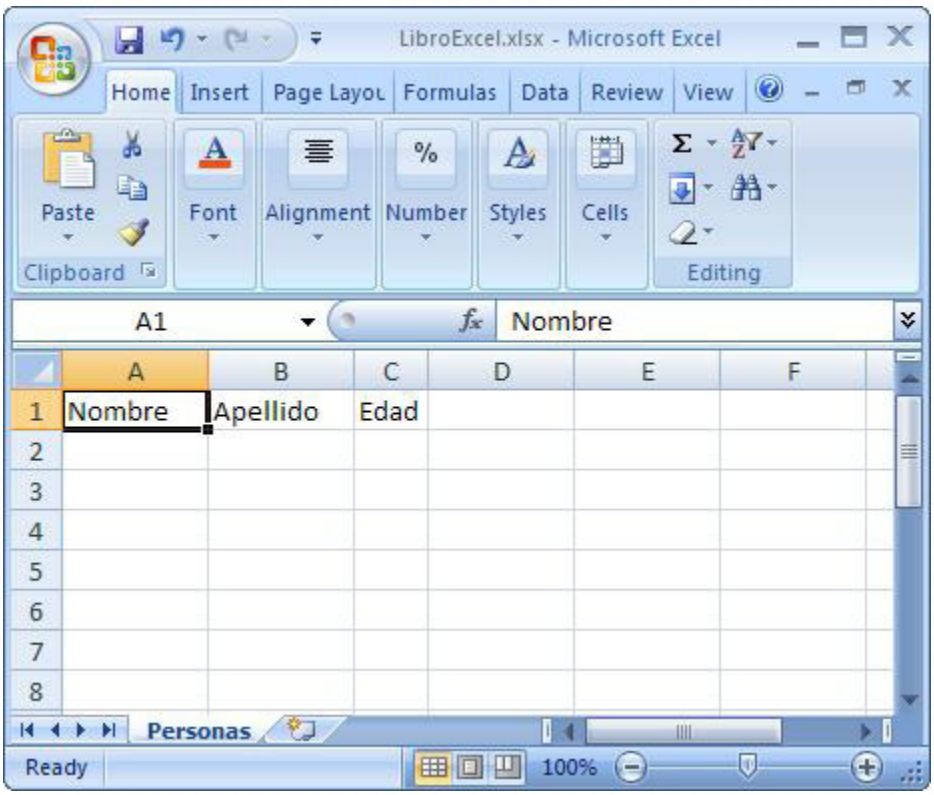
Acceso a datos, consultar y guardar información desde WebForms

Capítulo 7

Acceso a datos, consultar y guardar información desde WebForms

El acceso de datos es una de las funcionalidades más importantes y necesarias que debe tener una página web, ADO.NET nos ofrece librerías completas para trabajar con acceso a datos. Por lo general para guardar y leer información se utilizan bases de datos tales como SQL Server, MySQL, Oracle y demás.

Vamos a trabajar con un repositorio de datos más accesibles como Excel en donde vamos a guardas y leer información trabajándolo desde ASP.NET. Creamos un libro de Excel y lo renombramos “Personas” las primeras tres columnas tendrán los campos nombre, apellido y edad. El archivo le pondremos el nombre de LibroExcel.xlsx y lo guardamos en el raíz del disco C. C:\LibroExcel.xlsx



Utilizaremos el libro de Excel como base de datos, por ello, asegurate de crearlo correctamente. Cada hoja será una tabla y cada columna con título serán los campos.

Trabajando con datos desde ASP.NET

Creamos un WebForm llamado ADO.aspx y le colocamos:

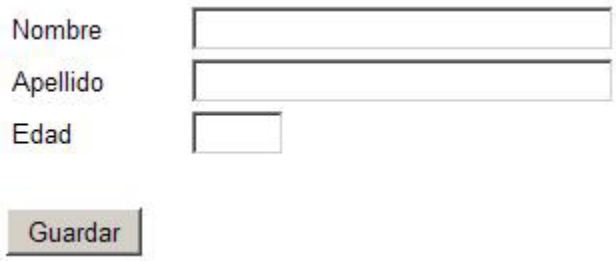
- 3 objetos: Label
- 3 objetos: TexBox
- 1 botón
- 1 GridView

C# Código fuente

```
<@ Page Language="C#" AutoEventWireup="true" CodeFile="ADO.aspx.cs"
Inherits="ADO"
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
<style type="text/css">
 .style1{width: 100%;}
 .style2{width: 86px; }
</style>
</head>
<body><form id="form1" runat="server">
<div>
<table class="style1">
<tr>
 <td class="style2">
  <asp:Label ID="lblNombre" runat="server" Font-Names="Arial" Font-Size="10pt"
Text="Nombre"></asp:Label>
 </td>
 <td>
  <asp:TextBox ID="txtNombre" runat="server" Width="205px"></asp:TextBox>
 </td>
</tr>
<tr>
 <td class="style2">
```

```
  <asp:Label ID="lblApellido" runat="server" Font-Names="Arial" Font-
Size="10pt" Text="Apellido"></asp:Label>
 </td>
 <td>
 <asp:TextBox ID="txtApellido" runat="server" Width="205px"></asp:TextBox>
 </td>
</tr>
<tr>
 <td class="style2">
  <asp:Label ID="lblEdad" runat="server" Font-Names="Arial" Font-Size="10pt"
Text="Edad"></sp:Label>
 </td>
 <td>
  <asp:TextBox ID="txtEdad" runat="server" Width="40px"></asp:TextBox>
 </td>
</tr>
<tr>
 <td class="style2">&amp;nbsp;</td>
 <td>
  <asp:Label ID="lblMensajes" runat="server" Font-Bold="True" Font-
Names="Arial"
 Font-Size="10pt" ForeColor="Red"></asp:Label>
 </td>
</tr>
</table>
<asp:Button ID="btnGuardar" runat="server" onclick="btnGuardar_Click"
Text="Guardar" /> <br />
<asp:GridView ID="GridView1" runat="server"></asp:GridView>
<br />
<br />
</div>
</form></body>
</html>
```

El formulario quedará de la siguiente forma:



La grilla aún no tiene datos por eso no se muestra en pantalla, este formulario solo tiene una acción y está asociada al botón:

```
<asp:Button ID="btnGuardar" runat="server"
onclick="btnGuardar_Click"
Text="Guardar" />
```

El botón guarda el contenido de los campos en la base de datos en Excel. Vamos a sumar más facilidades que nos provee .NET para el acceso a datos, utilizaremos la librería System.Data.OleDb que contiene objetos para conectar y ejecutar acciones sobre la conexión. Los objetos nuevos son:

using System.Data.OleDb;

- `OleDbConnection`
- `OleDbCommand`
- `OleDbDataAdapter`

using System.Data; DataTable

Veamos como se utilizan estos objetos de conexión en la práctica. Las funcionalidades básicas son dos: leer y escribir, entonces armaremos dos métodos para responder a estas necesidades.

Método Leer ():

Necesita una cadena de conexión, debemos decirle que proveedor de datos vamos a necesitar, donde está la fuente y de qué tipo es.

```
@"Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\\LibroExcel.xlsx; Extended
Properties=""Excel 12.0 Xml; HDR=YES;""";
```

Si no tienen este proveedor instalado lo pueden descargar gratis del centro de descarga de Microsoft (Access Database Engine). Luego, establecemos la conexión y levantamos datos del Excel. Por último bajamos los datos a un DataTable y lo enlazamos con nuestro GridView para ver los datos en pantalla.

C# Código fuente

```
private void Leer()
{
string connectionString = @"Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\\
LibroExcel.OleDbConnection conn = new OleDbConnection(connectionString);
OleDbDataAdapter da = new OleDbDataAdapter("Select * From [Personas$]", conn);
DataTable dt = new DataTable();
da.Fill(dt);
GridView1.DataSource = dt;
GridView1.DataBind();
}
```

Método Escribir ():

La forma de acceder a los datos es similar al leer con la diferencia que en lugar de usar un DataAdapter vamos a usar un DbCommand para insertar los nuevos datos en nuestra base.

C# Código fuente

```
private void Escribir()
{
string connectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\\
LibroExcel.OleDbConnection conn = new OleDbConnection(connectionString);
string insert = "Insert into [Personas$] (Nombre,Apellido,Edad) Values (?,?,?)"
OleDbCommand insertCommand = new OleDbCommand(insert, conn);
try
{
insertCommand.Parameters.Add("Nombre", OleDbType.VarChar).Value = txtNombre.
Text;
insertCommand.Parameters.Add("Apellido", OleDbType.VarChar).Value =
txtApellido.Text;
```

```
if (IsNumeric(txtEdad.Text))
{
insertCommand.Parameters.Add("Edad", OleDbType.Integer).Value = Convert.
ToInt32(txtEdad.}
else
{
throw new Exception("La edad debe ser numerica");
}
conn.Open();
int count = insertCommand.ExecuteNonQuery();
}
catch (OleDbException ex)
{
lblMensajes.Text = ex.Message;
}
catch (Exception ex)
{
lblMensajes.Text = ex.Message;
}
finally
{
conn.Close();
}
}
```

Ya tenemos los dos métodos más importantes ahora los llamaremos desde el botón (método escribir) y desde el Page_Load (evento leer).

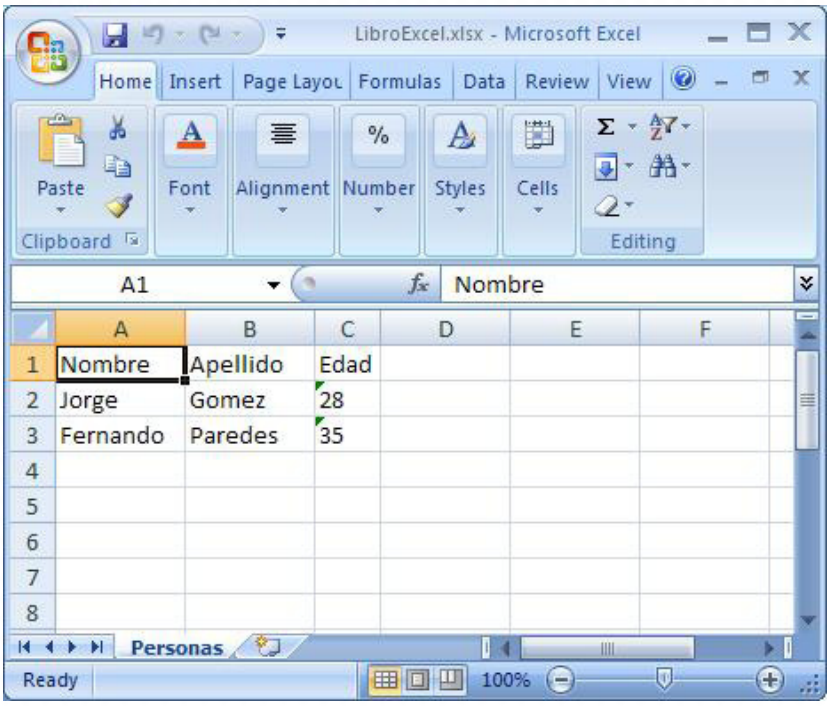
C# Código fuente

```
protected void Page_Load(object sender, EventArgs e)
{Leer();}
protected void btnGuardar_Click(object sender, EventArgs e)
{Escribir();}
```

Dentro del método Escribir estamos utilizando el método IsNumeric(txtEdad.Text) que vimos con anterioridad para comprobar si la edad ingresada es numérica. Así queda en funcionamiento nuestro primer contacto con conexión a datos. Cargamos datos en los campos y presionamos el botón Guardar.

Agregamos un nuevo registro.

Nuestro libro de Excel se verá de la siguiente forma:



Capítulo 8

Estructura de clases y objetos

Capítulo 8

Estructura de clases y objetos

La solución para reutilizar código y funcionalidad en nuestros proyectos es introducir código en la carpeta App_Code donde creamos clases con métodos que pueden utilizarse desde todo el proyecto y se compila en tiempo de ejecución. Puede organizar el código fuente de la forma que creamos más conveniente, ya que ASP.NET lo compilará en un sólo ensamblado al que tiene acceso el código de cualquier parte de la aplicación web.

Código compartido

En los ejemplos anteriores necesitamos de código compartido. Por ejemplo, el método IsNumeric lo utilizamos en el “Controles de usuarios reutilizables” y “Acceso a datos”.

- Estructura de una clase: una clase está compuesta por Atributos, Propiedades, Métodos y Eventos. Aunque no necesariamente deba tener todos sus componentes declarados, podemos crear clases que sólo tengan métodos o que sólo tengan atributos y propiedades.
- Ejemplo de una clase básica: el primer evento de nuestra clase será el constructor, al instanciar la clase (Objeto) lo primero que se ejecuta es el constructor.

C# Código fuente

```
public class Herramientas
{
 public Herramientas()
 {}
}
```

Ahora agregaremos un método que nos facilite la reutilización de código, como por ejemplo: *IsNumeric()*.

C# Código fuente

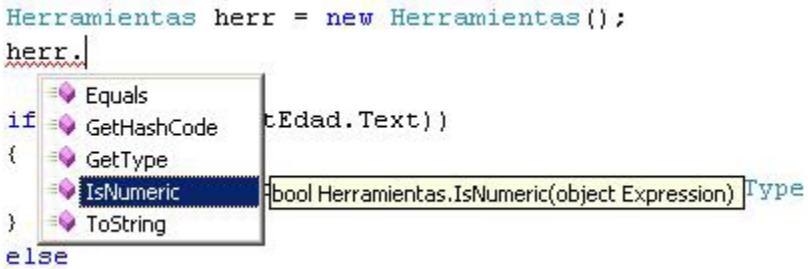
```
using System;
using System.Collections.Generic;
using System.Web;
public class Herramientas
{
public Herramientas()
{
}
public bool IsNumeric(object Expression)
{
bool isNum;
double retNum;
isNum = Double.TryParse(Convert.ToString(Expression), System.Globalization.
NumberSty
return isNum;
}
}
```

Ya tenemos creada nuestra clase Herramientas.cs en la carpeta App_Code. ¿Cómo podemos utilizar el método IsNumeric() desde cualquier parte de nuestra solución Web?

Lo primero que debemos hacer es instanciarla.

```
Herramientas herr = new Herramientas();
```

Vemos dentro de nuestro objeto herr los métodos públicos.



Este método nos devuelve un Booleano: verdadero o falso y nos sirve para evaluar. Vamos a llamar a la clase Herramientas desde el método Escribir del proceso anterior.

```
Herramientas herr = new Herramientas();
if (herr.IsNumeric(txtEdad.Text))
{
}
```

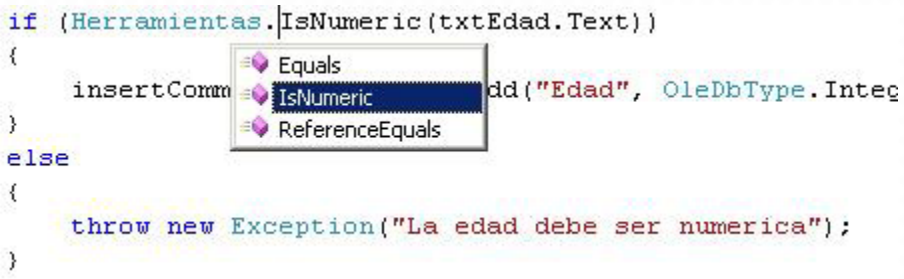
De esta forma nos evitamos codificar el evento IsNumeric en cada parte del código donde lo necesitemos.

Clases estáticas:

Son clases que no tienen un constructor invocable, por consiguiente no hay que instanciarlas y se pueden acceder a los métodos en forma directa.Veamos el ejemplo anterior pero declarando la clase como estática y su utilización.

```
Public static class Herramientas
{
 static Herramientas()
 {}
 public static bool IsNumeric(object Expression)
 {}
}
```

Al no tener la necesidad de instanciarla la llamada al método IsNumeric lo hacemos en forma directa de la siguiente forma.



Capítulo 9

Utilización de Master Pages

Capítulo 9

Utilización de Master Pages

Cuando creamos un sitio web tenemos la necesidad de repetir ciertas partes de una página en todo el sitio o en parte del sitio. Para no estar copiando y pegando las mismas estructuras en todas las páginas lo que podemos hacer es crear una Master Page y referenciarla en las otras páginas.

¿Cómo lo hacemos?

La construcción de las páginas maestras son similares a la creación de web forms con algunas diferencias.

- La extensión del archivo es .master
- Usa la directiva @ Master
- Contiene un objeto ContentPlaceHolder donde lo utilizara para mostrar el contenido de las páginas del sitio.

Veamos la estructura de una Master Page: la llamaremos Main.master

C# Código fuente

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="Main.master.cs"
Inherits=
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Label ID="Label1" runat="server" Text="Mi sitio Web para Maestros del Web"
Font-Names="Tahoma" Font-Size="14pt"></asp:Label>
<br />
```

```
<asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
</div>
</form></body></html>
```

Al igual que los web forms las master pages puede tener su código separado *CodeFile="Main.master.cs"*

En nuestro ejemplo vemos que hemos agregado al formulario un Label con un texto:

Ya tenemos creada la master page con su ContentPlaceHolder1 donde se mostrará el contenido del web form. Para indicarle a un web form que deseamos este contenido en la master page debemos agregar la propiedad MasterPageFile="~/Main.master" a la directiva page.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Main.master"
AutoEventWireup="true"
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content&gt;
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server"
<%--Aqui dentro van los objetos de nuestro web forma--%>
</asp:Content>
```

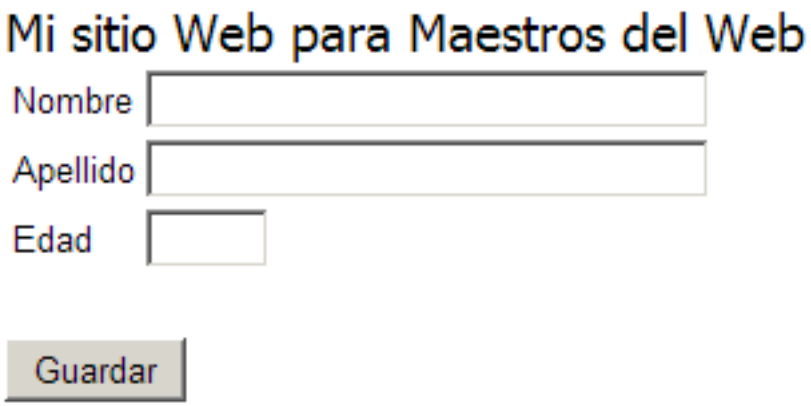
Como vemos la estructura del aspx debe cambiar, ya los tags html no están porque ya están en la master page, entonces sólo debemos ubicar los controles dentro de ContentPlaceHolder1 de la siguiente forma.

Tomamos como ejemplo ADO.aspx del capítulo anterior, y queda de la siguiente forma:

```
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server"
<table>
<tr><td>
<asp:Label ID="lblNombre" runat="server" Font-Names="Arial" Font-Size="10pt"
Text="Nombre"></asp:Label>
</td>
<td>
<asp:TextBox ID="txtNombre" runat="server" Width="205px"></asp:TextBox>
</td>
</tr>
<tr><td>
<asp:Label ID="lblApellido" runat="server" Font-Names="Arial" Font-Size="10pt"
Text="Apellido"></asp:Label>
</td>
<td>
<asp:TextBox ID="txtApellido" runat="server" Width="205px"></asp:TextBox>
</td>
</tr>
<tr><td>
<asp:Label ID="lblEdad" runat="server" Font-Names="Arial" Font-Size="10pt"
Text="Edad"></asp:Label>
</td>
<td>
<asp:TextBox ID="txtEdad" runat="server" Width="40px"></asp:TextBox>
</td>
</tr>
<tr><td>
&amp;nbsp;</td>
<td>
```

```
<asp:Label ID="lblMensajes" runat="server" Font-Bold="True" Font-Names="Arial"
Font-Size="10pt" ForeColor="Red"></asp:Label>
</td>
</tr>
</table>
<asp:Button ID="btnGuardar" runat="server"
Text="Guardar" />
<br />
<asp:GridView ID="GridView1" runat="server">
</asp:GridView>
</asp:Content>
```

Si ejecutamos ahora nuestro ADO.aspx lo veremos de la siguiente manera.



Donde el título y los tags HTML están en la master page y los controles están en ADO.aspx

Cómo crear menús de navegación

Podemos crear menús de navegación para su sitio web utilizando el control menú de ASP.NET. El control menú tiene ciertas propiedades que nos permiten adaptar a nuestro gusto y necesidad. Por default el control menú es un control vertical. Para nuestra página vamos a necesitar que sea horizontal.

```
Orientation="Horizontal"
Orientation="Vertical"
```


```
<asp:Menu ID="Menu1" runat="server"></asp:Menu>
```

Todo menú debe tener ítems que nos sirvan de acceso a las diferentes partes del sitio.

```
<Items></Items>
```

Dentro de los tags ítems debemos detallar los MenuItems que necesitamos.

```
<asp:MenuItem></asp:MenuItem>
```

Los menús se comportan en relación a una estructura de árboles normalmente organizados en diferentes niveles de una jerarquía. La primera rama es el root del menú y las siguientes son los sub menú.

```
<asp:MenuItem>
<asp:MenuItem>
</asp:MenuItem>
<asp:MenuItem>
<asp:MenuItem />
<asp:MenuItem />
<asp:MenuItem />
</asp:MenuItem>
</asp:MenuItem>
```

Creamos un menú con tres opciones para nuestro sitio web y lo vamos a incluir en la master page.

C# Código fuente

```
<asp:Menu ID="Menu1" runat="server" />
<Items>
```

```
<asp:MenuItem NavigateUrl="~/Menu1.aspx" Text="Menu1" Value="1"/>
<asp:MenuItem NavigateUrl="~/Menu2.aspx" Text="Menu2" Value="2"/>
<asp:MenuItem NavigateUrl="~/Menu3.aspx" Text="Menu3" Value="3"/>
</Items>
</asp:Menu>
```

ASP.NET permite agregarle estilo al control mediante determinadas propiedades del tab MENU.

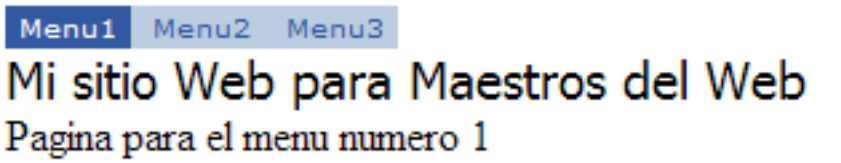
C# Código fuente

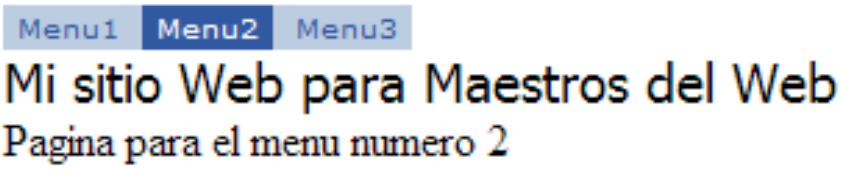
```
BackColor="#B5C7DE"
DynamicHorizontalOffset="2"
Font-Names="Verdana"
Font-Size="0.8em"
ForeColor="#284E98"
StaticSubMenuIndent="10px"
<StaticSelectedStyle BackColor="#507CD1" />
<StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px" />
<DynamicHoverStyle BackColor="#284E98" ForeColor="White" />
<DynamicMenuStyle BackColor="#B5C7DE" />
<DynamicSelectedStyle BackColor="#507CD1" />
<DynamicMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px" />
<StaticHoverStyle BackColor="#284E98" ForeColor="White" />
```

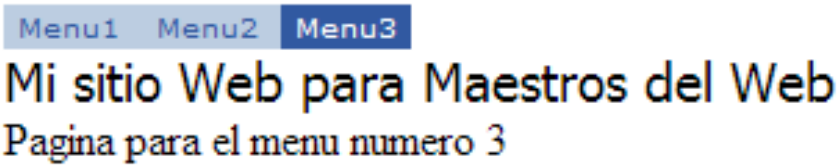
.El menú y formato se ve de la siguiente forma:

```
Orientation="Vertical"
```


```
Orientation="Vertical"
```







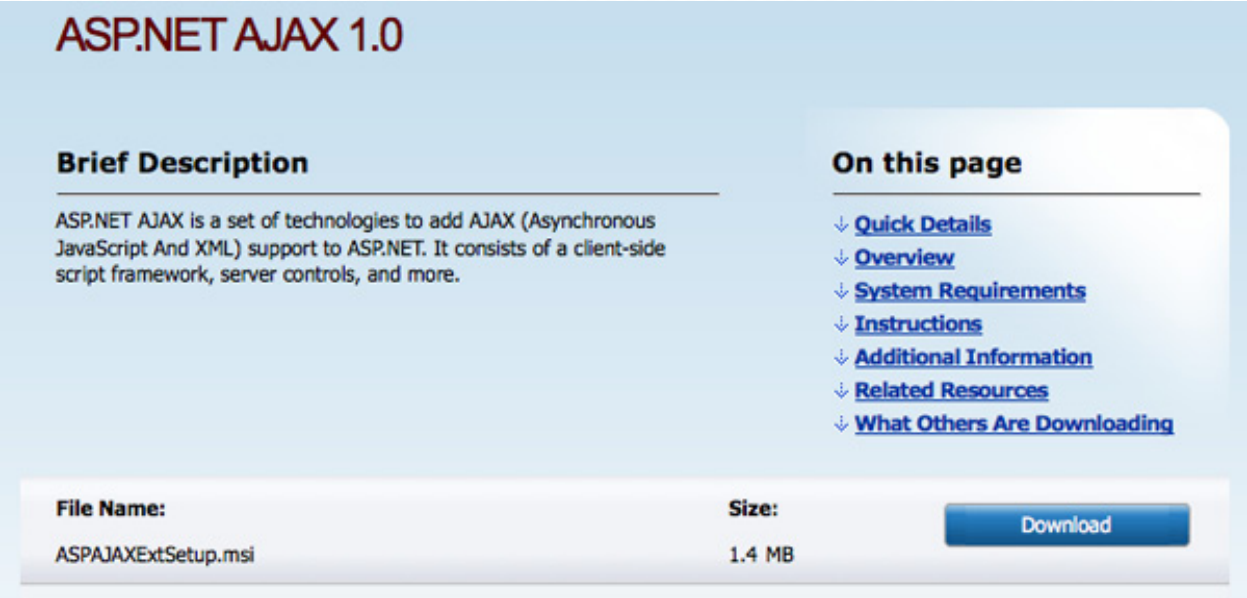
Capítulo 10

Utilización de Ajax

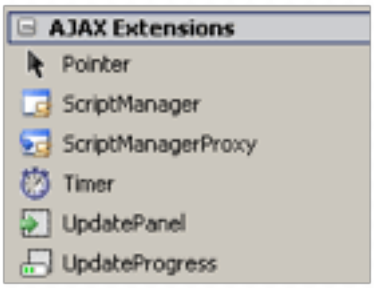
Capítulo 10

Utilización de Ajax

ASP.NET/Ajax es un conjunto de extensiones que permiten la implementación de Ajax en web forms que nos permiten actualizar datos en la pantalla sin necesidad de una recarga completa de la misma, para usarlo en ASP.NET es necesario tener instaladas dichas extensiones. Podemos encontrarlas en el centro de descargas de Microsoft:



Una vez instaladas funcionarán los controles AJAX en nuestros formularios ASP.NET



El control AJAX más usado es el UpdatePanel que justamente nos permite recargar solo una porción de nuestra página. El control UpdatePanel necesita del ScriptManager para funcionar.

¿Cómo se implementa Ajax en nuestro sitio Web?

Una vez instaladas las extensiones de Ajax debemos indicarle a nuestro sitio web que deberá responder a controles Ajax. Esto lo hacemos agregando las siguientes líneas en nuestro Web.Config

C# Código fuente

```
System.Web.Extensions
System.Web.Extensions.Design
<runtime>/code>
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
<dependentAssembly>
<assemblyIdentity name="System.Web.Extensions" publicKeyToken="31bf3856ad3
64e35"
<bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0"/>
</dependentAssembly>
<dependentAssembly>
<assemblyIdentity name="System.Web.Extensions.Design" publicKeyToken="31bf3856a
d364e35"
<bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0"/>
</dependentAssembly>
</assemblyBinding>
</runtime>
```

En caso de utilizar la IDE del Visual Studio no sería necesario agregar estas líneas ya que al crear el proyecto las agrega por default. Uno de los formularios que realizan postback, es decir, que van al servidor y se vuelven a recargar es el formulario ADO.NET. En este vamos a implementar el UpdatePanel para que no recargue toda la página.

Agregamos el control ScriptManager en la parte inicial del webform o del ContentPlaceHolder. En nuestro caso que estamos usando Master Page con el webform ADO.aspx.

```
<asp:ScriptManager ID="ScriptManager1" runat="server"> </asp:ScriptManager>
```

Luego colocamos todo el formulario que creamos (cuadros de textos, labels, botons, grilla, tablas, etc.) dentro del UpdatePanel.

C# Código fuente

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
<ContentTemplate>
…Aquí pondremos todos los controles
</ContentTemplate>
</asp:UpdatePanel></code>
```

Ejecutamos ADO.aspx y veremos que cada vez que guarda o lee información no va a recargar toda la página, solamente la porción que está dentro del UpdatePanel.

Otras guías

iPhone

Crea tu propia aplicación para dispositivos móviles basados en iOS, y descubre las posibilidades de trabajo que este mercado ofrece a los desarrolladores.

Visita la Guía iPhone

Zend

Zend Framework es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5.

Visita la Guía Zend

Producción de Video

Tendencias en la creación de videos, herramientas, consejos y referencias en la producción de videos para la web.

Visita la Guía Producción de vídeo

Curso Android

Actualiza tus conocimientos con el curso sobre Android para el desarrollo de aplicaciones móviles.

Visita el curso Android

Mapas

Aprende a utilizar el API de Google Maps para el desarrollo de tus proyectos con mapas.

Visita la Guía Mapas

Startup

Aprende las oportunidades, retos y estrategias que toda persona debe conocer al momento de emprender.

Visita la Guía Startup

Adictos a la comunicación

Utiliza las herramientas sociales en Internet para crear proyectos de comunicación independientes.

Visita Adictos a la comunicación