



ASP.NET 3.5

FUNDAMENTOS

CLIPSA



DISTRIBUIDO POR:

CENTRO DE INFORMÁTICA PROFESIONAL S.L.

C/ URGELL, 100
08011 BARCELONA
TFNO: 93 426 50 87

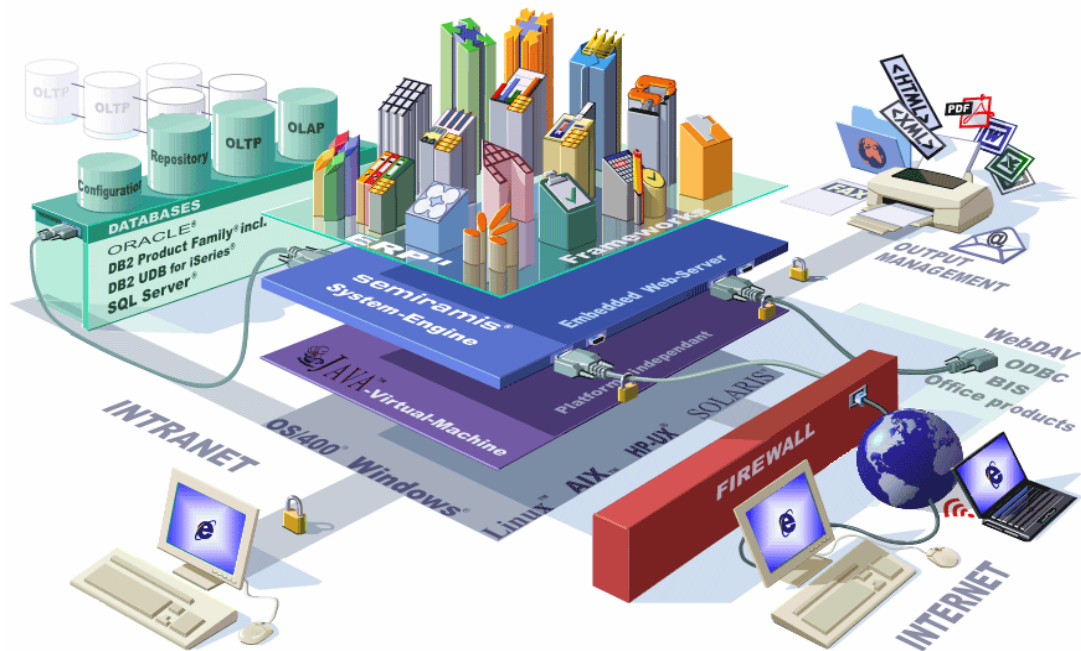
C/ RAFAELA YBARRA, 10
48014 BILBAO
TFNO: 94 448 31 33

www.cipsa.net

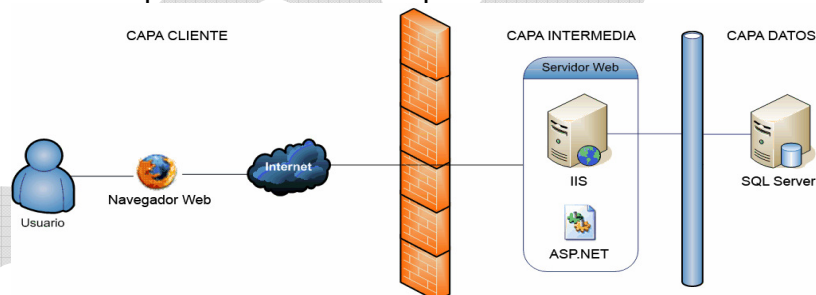
RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR

CLIPSA

Aplicaciones Web



Una aplicación Web es un programa informático que permite dar servicio a múltiples usuarios a través de Internet. Estas aplicaciones se basan en lo que se conoce como “arquitectura de tres capas”:



1.1.- LA CAPA CLIENTE

Tiene dos funciones:

1.- Capturar los datos necesarios de usuario con los que opera la aplicación y enviárselos a la misma. Suponiendo una aplicación de búsqueda de números de teléfono, la captura de datos consistiría en un formulario que solicita al usuario el nombre y apellidos de la persona cuyo número quiere localizarse.

2.- Representar la información de respuesta de la aplicación para el usuario. Suponiendo la aplicación antes mencionada, la respuesta consistiría en mostrar una tabla con los nombres y teléfonos de las personas coincidentes con los datos introducidos por el usuario.

La capa cliente de una aplicación Web se ejecuta en el navegador Web del usuario empleando el protocolo *HTTP* y páginas *HTML*.

1.1.1.- NAVEGADORES WEB

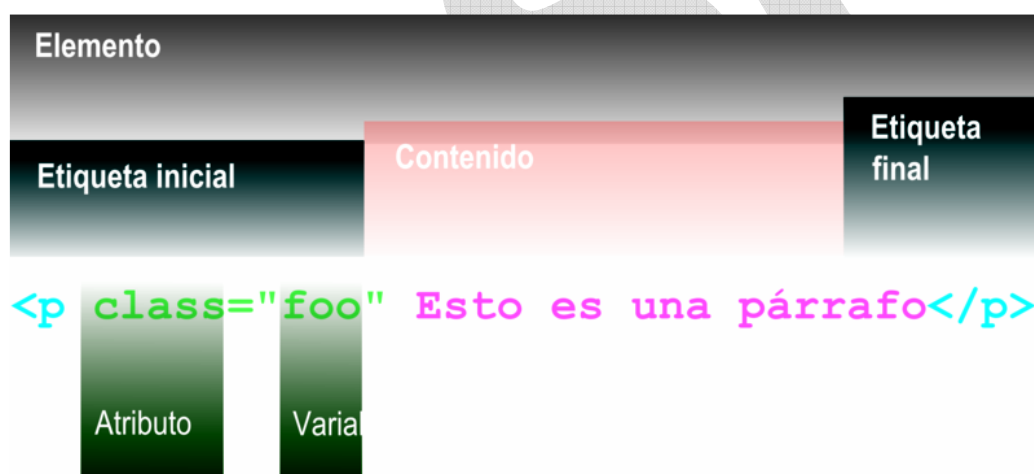
El navegador Web es un programa instalado en el equipo del cliente (un ordenador convencional, una PDA, o un teléfono móvil), que permite navegar e interactuar con páginas HTML. La comunicación entre la aplicación Web y el navegador del cliente se lleva acabo empleando el protocolo HTTP.

Entre los más extendidos actualmente se encuentran el Microsoft Internet Explorer 7.0, y el Mozilla FireFox 3.0.

1.1.2.- PAGINAS WEB Y HTML

Una página Web es básicamente un documento de texto que contiene información que es interpretada y representada por un navegador Web. Las páginas Web se codifican mediante HTML.

El HTML es un lenguaje basado en etiquetas que describe la estructura y contenido de las páginas Web. Estas etiquetas se componen de atributos (que identifican cómo mostrar los datos), y contenido (que identifica los datos mismos).



Actualmente se tiende a emplear cada vez más el XHTML («lenguaje extensible de marcado de hipertexto»).

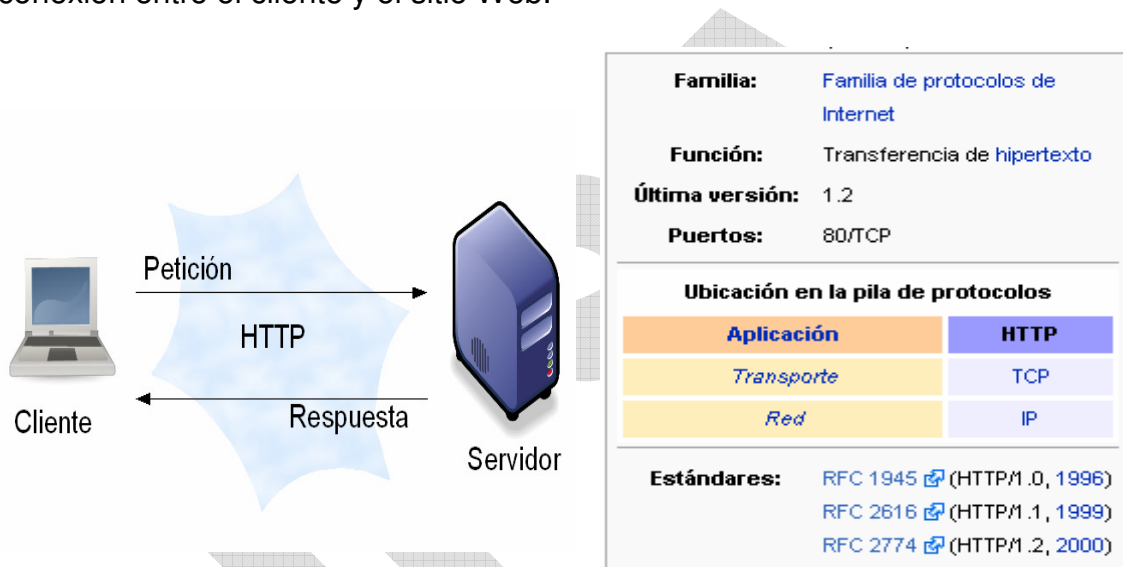
XML, sigla en inglés de *Extensible Markup Language* («lenguaje de marcas ampliable»), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades, como es el caso de XHTML.

El XHTML está pensando para sustituir a HTML como lenguaje descriptor para las páginas Web. La primera versión de XHTML es una mera adaptación de HTML a XML que posee las funcionalidades del HTML original pero cumpliendo las especificaciones de XML en su expresión.

1.1.3.- EL PROTOCOLO HTTP

HTTP (*Protocolo de Transferencia de Hipertexto*) es un protocolo empleando en las transacciones Web para la comunicación entre navegadores y sitios Web. Este protocolo se apoya a su vez en el protocolo de comunicaciones TCP/IP que se emplea a nivel global en las comunicaciones en Internet.

Inicialmente HTTP solo permitía la transacción exclusiva de texto entre navegador y servidor Web mediante peticiones y respuestas. La petición HTTP permite a un navegador Web solicitar a un sitio Web una determinada página. El servidor busca la página solicitada y la envía al cliente Web correspondiente mediante una respuesta HTTP. Una vez completada la transmisión, se cierra la conexión entre el cliente y el sitio Web.



Ejemplo: Petición HTTP de un navegador Web solicitando al servidor Web la página /localhost/hola.html.

```
GET /localhost/hola.htm
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, .... , */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; windows NT 5.1m ... .NET CLR 3.0.04506.30 )
Host: localhost:80
Connection: Keep-Alive
```

Ejemplo: Respuesta HTTP del servidor Web, con el código HTML de la página solicitada embebida.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
X-Powered-By: ASP.NET
Date: Thu, 01 Nov 2007 17:54:34 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Mon, 02 Nov 2007 17:45:56 GMT
ETag: "04e9ace185fc51:bb6"
Content-Length: 130
<html>                                     ← A partir de aquí empieza el HTML
  <body>
    <h1>Hello world</h1>
  </body>
</html>
```

1.2.- LA CAPA INTERMEDIA

La capa intermedia contiene la propia aplicación Web instalada en un servidor Web al que acceden los usuarios a través de red o de Internet.

Las funciones de la capa intermedia son:

1. Recibir las peticiones y datos enviados por los navegadores de los.
2. Procesar las peticiones e información recibida desde el usuario, accediendo a las bases de datos necesarias.
3. Enviar las páginas con la información de respuesta al usuario correspondiente.

La capa intermedia se ejecuta en los servidores Web. Un servidor Web es un ordenador que posee instalado un software que atiende peticiones HTTP desde los navegadores de los clientes y responde con la página solicitada

Actualmente los servidores Web más conocidos son el *Internet Information Services* de Microsoft (también llamado *IIS*) , y el *Apache* de la *Apache Software Foundation*.

1.2.1.- EL INTERNET INFORMATION SERVICES (IIS)

Todos los entornos de aplicación Web trabajan fundamentalmente de la misma manera. Se requiere un servicio Web que escucha un determinado puerto de un equipo con una determinada dirección IP. El puerto TCP/IP empleado para el envío y atención de peticiones HTTP es normalmente el 80.

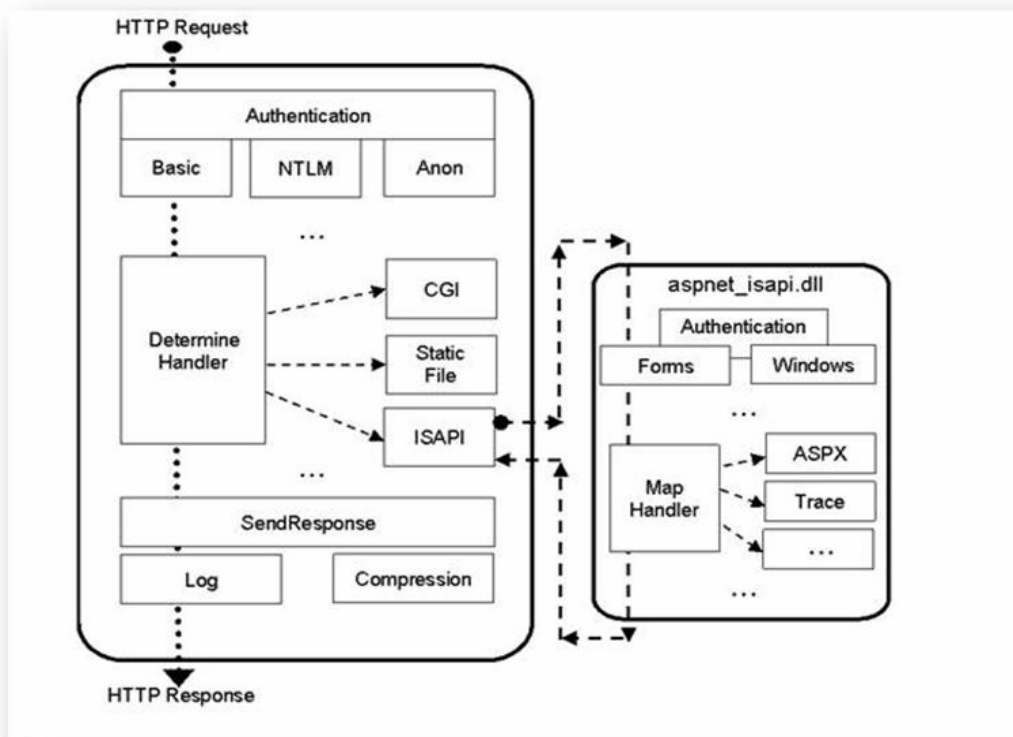
Normalmente, los servicios Web se instalan y mantienen funcionando en equipos específicos para ello, en cuyo caso; tales equipos son denominados Servidores Web. En muchas ocasiones los términos servicio Web y servidor Web se intercambian entendiendo que el Servidor Web es el equipo en el que se ejecuta el Servicio Web.

Por defecto, cuando un navegador Web solicita una página a un servidor Web IIS, éste la intercepta y la procesa. El procesado de una petición HTTP (*HTTP Request*), conlleva identificar el tipo de recurso que se solicita, (una imagen, una página web, una página ASP), y la ejecución de la librería ISAPI correspondiente para atender la petición. En el caso de solicitarse una página de ASP.NET, la librería encargada de procesar la solicitud se llama *aspnet_isapi*.

La atención de una petición HTTP lleva asociado normalmente el generar una respuesta HTTP (*HTTP Response*) que transporte la información requerida en la petición. En la mayoría de los casos esta respuesta suele ser el código de una página Web cuya dirección ha sido solicitada previamente por una petición. Este código puede corresponderse con el de una página estática almacenada en el servidor, o ser el resultado de la ejecución de una página ASP.NET.

Si se da el caso que la página o imagen solicitadas no están disponibles, se envía igualmente una respuesta HTTP con un código de error.

El siguiente esquema muestra el flujo de procesamiento de una solicitud a un servidor IIS desde la recepción de la petición de una página por parte del cliente (*HTTP Request*), hasta el envío de la página solicitada como respuesta (*HTTP Response*).



Esquema de funcionamiento de Internet Information Server

1.2.2.- FUNDAMENTOS DE LAS PAGINAS DINAMICAS

Por definición una página dinámica es aquella cuyo contenido no está prestablecido desde el principio al completo. Las páginas dinámicas contienen secciones que se generan al ser solicitadas en función de datos enviados por el usuario, o almacenados en ficheros o bases de datos en el servidor.

Al principio las aplicaciones Web se programaban en C/C++ creando versiones modificadas de la propia librería ISAPI que atendieran las peticiones HTTP entrantes, y generasen las correspondientes respuestas. Sin embargo, escribir un sitio completo de esta manera supone bastante complejidad a pesar de que confiere control total sobre el comportamiento Web.

ASP fue el primer intento de crear una librería ISAPI única que permitiera crear aplicaciones Web empleando lenguajes de script como javascript o vbscript. Esta librería recibió el nombre de ASP.DLL y procesaba páginas con extensión .asp, que contenían HTML mezclado con scripts ejecutables. Al solicitarse una página asp, la librería ASP.DLL ejecutaba los scripts de la página e incrustaba los resultados en el HTML de la página. Finalmente la página era enviada al usuario en una respuesta HTTP.

Supóngase el código de la siguiente página almacenada en nuestro sitio Web con el nombre `hola.asp`.

```
<%@ Language="javascript" %>
<html>
<body>
<h3>Esto es una pagina de ASP</h3>
<h1><% Response.Write("Este contenido va a ser ejecutado"); %></h1>
</body>
</html>
```

Cuando un navegador solicita al sitio Web esta página, el IIS reconoce la extensión de la página (`.asp`), y la desvía a la librería `ASP.DLL` para su procesamiento.

La primera línea indica el lenguaje de script empleado, en el caso de este ejemplo, se trata de JavaScript. La librería recorre el fichero HTML sin hacer nada hasta llegar a la línea delimitada por las etiquetas `<%..%>` que indican la presencia una sección de código a ejecutar en el servidor Web.

En el ejemplo, el código de servidor es una llamada al objeto *Response* para que escriba la cadena “Este contenido va a ser ejecutado” en el fichero respuesta. El código se ejecuta y la cadena generada es insertada en su lugar dentro del HTML. Finalmente el código HTML resultante es enviado al navegador cliente:

```
<html>
<body>
<h3>Esto es una pagina de ASP</h3>
<h1>Este contenido va a ser ejecutado</h1>
</body>
</html>
```

El código HTML resultante no tiene ni rastro de scripts debido a que éste se ejecutan en el servidor Web, y el navegador cliente sólo recibe el HTML resultante.

Supongamos el siguiente código de la página: “*form.asp*”

```
<%@ Language="javascript" %>
<html>
<body>
<form action="form.asp" method="POST">
<h2>Cual es tu animal favorito?</h2>
<select name="lista">
<option>Perro</option>
<option>Gato</option>
<option>Tortuga</option>
</select>
<input type="submit" name="enviar" value="enviar" />
</form>
<br />
<p>Seleccionastes: <%=Request("Lista")%></p>
</body>
</html>
```

La página contiene el código HTML de un formulario con una lista desplegable de tres elementos “Perro”, “Gato”, y “Tortuga”. Después del formulario hay una línea de código de servidor `<%=Request("Lista")%>` que muestra el valor del elemento seleccionado en la lista del formulario.

Si solicitamos la página *form.asp* obtendremos la siguiente respuesta:

```
<html>
<body>
<form action="form.asp" method="POST">
<h2>Cual es tu animal favorito?</h2>
<select name="lista">
<option>Perro</option>
<option>Gato</option>
<option>Tortuga</option>
</select>
<input type="submit" name="enviar" value="enviar" />
</form>
<br />
<p>Seleccionastes: </p>
</body>
</html>
```

Cual es tu animal favorito?

Seleccionastes:

El HTML de respuesta muestra que el código de servidor no ha devuelto ningún resultado, puesto que la página se había solicitado por primera vez y no recibió ninguna información en la petición.

Supongamos que ahora seleccionamos el último elemento de la lista "Tortuga" y pulsamos el botón enviar. El atributo action de la etiqueta `<form>` indica que los datos se envían de vuelta contra la propia página "form.asp". El resultado que obtenemos como respuesta ahora es:

```
<html>
<body>
<form action="form.asp" method="POST">
<h2>Cual es tu animal favorito?</h2>
<select name="lista">
<option>Perro</option>
<option>Gato</option>
<option>Tortuga</option>
</select>
<input type="submit" name="enviar" value="enviar" />
</form>
<br />
<p>Seleccionastes: Tortuga</p>
</body>
</html>
```

Cual es tu animal favorito?

Seleccionastes: Tortuga

En esta ocasión, el código de servidor ha devuelto efectivamente el elemento que habíamos seleccionado en la lista del formulario, y cuyo valor fue enviado al servidor como parte de la petición al pulsar el botón "enviar".

Sin embargo, la lista no refleja ya el valor seleccionado debido a que el control no mantiene el estado de selección de la página anterior. Esta es una de las grandes diferencias que existen entre el ASP, y el ASP.NET. En ASP.NET el estado de los controles se mantiene gracias al uso de controles Web de servidor.

1.3.- LA CAPA DATOS

La capa de datos se encarga del almacenamiento permanente de la información manejada por la aplicación Web. Para ello se emplean normalmente sistemas basados en bases de datos relacionales.

Una base de datos relacional consiste en un conjunto de tablas relacionadas entre si. La información a almacenar se organiza entre las diferentes tablas que componen la base de datos, manteniendo su integridad y facilitando el acceso a cualquier dato.

La capa de datos se ocupa de manejar una o varias bases de datos a través de un servidor de bases de datos. Los servidores de bases de datos son equipos dotados de un software de gestión que administra las bases de datos y permite que múltiples usuarios o aplicaciones puedan conectarse y utilizarlas al mismo tiempo. Actualmente los servidores de bases de datos más conocidos son Microsoft SQL Server, MySQL y Oracle entre otros.

Características de ASP.NET

2.1.- ANATOMIA DE UNA APLICACIÓN WEB ASP.NET

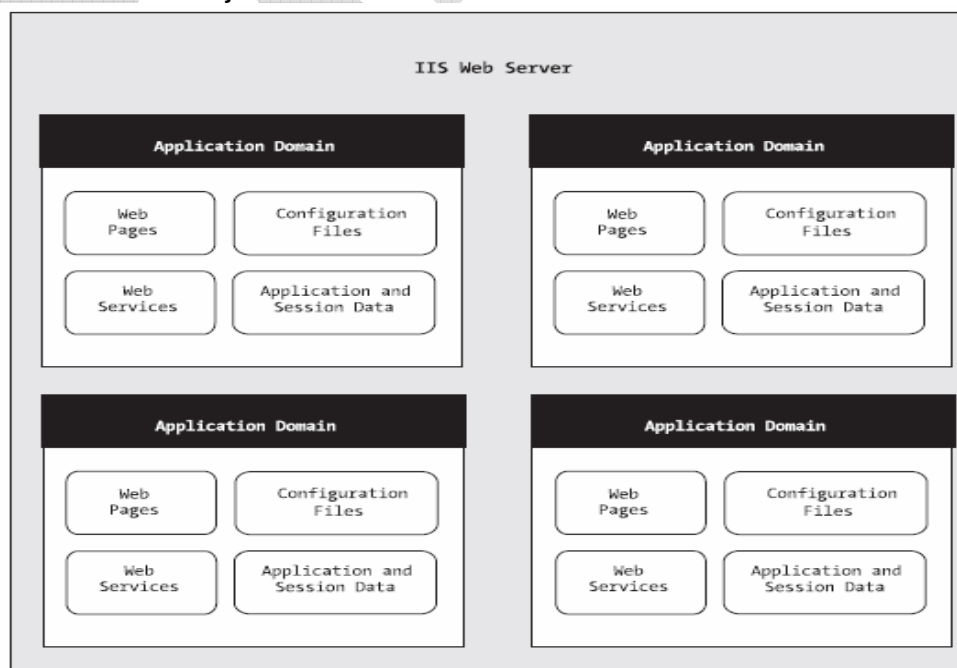
A diferencia de las aplicaciones de escritorio que se ejecutan a partir de un archivo ejecutable .exe; las aplicaciones Web se componen de múltiples páginas web. Un usuario puede por lo tanto acceder a una aplicación Web a través de diferentes páginas o siguiendo un hipervínculo desde otra página externa a la aplicación Web.

Aunque una aplicación Web esté formada por varias páginas independientes, no obstante; todas aquellas que forman parte de la misma aplicación Web en ASP.NET comparten y tienen acceso a una serie de recursos y archivos de configuración. Estos archivos y recursos son exclusivos de una aplicación Web y no se comparten entre aplicaciones ni siquiera aunque estén alojadas en el mismo servidor Web.

Cada aplicación Web ocupa un determinado espacio en la memoria del servidor Web que se mantiene y gestiona de forma independiente cada una. Esto impide que los datos de varias aplicaciones se mezclen y que si una aplicación falla detenga todo el servidor.

La composición de una aplicación Web ASP.NET se define como un conjunto de páginas, archivos, ficheros de código ejecutables agrupados en un directorio virtual. Un directorio virtual se corresponde con una carpeta en el disco duro del servidor Web cuyo contenido se expone públicamente y pueden solicitarse desde un navegador Web.

La siguiente imagen muestra el esquema de un servidor Web con diferentes aplicaciones Web alojadas en diferentes directorios virtuales:



Aplicaciones Web en un servidor IIS

2.1.1.- Archivos de una aplicación ASP.NET

Una aplicación Web ASP.NET está compuesta por los siguientes tipos de archivos. Todos ellos pueden distinguirse por su extensión:

- *.aspx* → Páginas Web ASP.NET. Estas páginas contienen el diseño de la página y opcionalmente código de servidor embebido. Las peticiones de los usuarios van dirigidas normalmente a estos archivos por lo que constituyen el punto de inicio de la aplicación Web.
- *.ascx* → Controles de usuario de ASP.NET. Los controles de usuario son semejantes a páginas Web pero no son accesibles directamente por los clientes. Los controles Web permiten reunir uno o varios controles Web en un mismo elemento y reutilizarlo en múltiples páginas Web.
- *.asmx* → Servicios Web ASP.NET. Los servicios Web son conjuntos de métodos que pueden ser invocados a través de red para su uso en aplicaciones Web o aplicaciones convencionales.
- *web.config* → Archivo de configuración de la aplicación Web. Este archivo está escrito en XML y configura opciones de seguridad, mantenimiento del estado, gestión de la memoria,... etc.
- *global.asax* → Archivo global de aplicación que permite definir variables globales accesibles desde cualquier página de una aplicación Web, e implementar eventos globales tales como el inicio de una aplicación Web, o el suceso de un error.
- *.cs* → Archivos de código ejecutable por el servidor Web asociado a una página Web ASP.NET. Estos archivos contienen la lógica de funcionamiento de la página mientras que el archivo *.aspx* contiene el diseño.

Adicionalmente una aplicación Web puede contener archivos de otros tipos tales no pertenecientes a ASP.NET, tales imágenes, páginas HTML, hojas de estilos CSS, scripts de Javascript,... etc.

2.1.2.- Carpeta de una aplicación ASP.NET

Una aplicación Web correctamente diseñada debe poseer una estructura de directorios en los que se organicen los distintos archivos que la componen de forma ordenada. Estas carpetas pueden crearse libremente por el desarrollador y organizados como estime oportuno.

Además de éstas, ASP.NET define una serie de carpetas destinadas a contener determinados archivos

- *Bin* : Contiene ensamblados de código ya compilado que implementan componentes o controles utilizado en la aplicación Web.
- *App_Code*: Contiene archivos de código fuente *.cs* de componentes o controles utilizados en la aplicación Web.
- *App_Data*: Almacena ficheros de datos tales como ficheros XML, ficheros de bases de datos...etc.
- *App_Themes*: Almacena temas que son empleados para gestionar el aspecto general de las páginas.

2.2.- CARACTERISTICAS DE ASP.NET 2.0

El ASP.NET proviene del ASP clásico, y mantienen ciertas semejanzas sintácticas a la hora de escribir el código. Muchos de los símbolos y etiquetas que se emplean en HTML para incrustar el código *script* ejecutable en ASP.NET son idénticos a los de ASP. Algunas de sus características más importantes son:

1.- Utilización de lenguajes .NET

ASP .NET soporta diversos lenguajes de programación, entre los que se encuentra C# y Visual Basic .NET. En este texto vamos a emplear C#, ya que es el nuevo lenguaje que ofrece Microsoft y pretende ser el lenguaje estándar para el .NET Framework.

2.- Separación de código y representación

Las páginas ASP.NET se dividen en dos archivos independientes. El primero archivo de ellos (.aspx) contiene la información del aspecto y controles que componen la página.

El segundo archivo (.aspx.cs en C#, y.aspx.vb en VB.NET), contiene el código de servidor que debe ejecutarse al producirse los distintos eventos de los controles contenidos en la página.

3.- Programación Orientada a Objetos basada en Eventos

En ASP .NET cada página es representada por una clase que hereda a su vez de la clase *Page* definida en el espacio de nombres System.Web.UI. Cada página posee atributos, métodos y eventos heredados de la clase Page y otros propios definidos por el programador.

De igual manera, los controles Web utilizados en las páginas Web están representados por clases heredadas en su mayoría de la clase *WebControl*, de donde heredan atributos, propiedades y eventos comunes a todos los controles a parte de los específicos de cada control.

2.3.- COMPONENTES DE ASP.NET

La tecnología ASP.NET ofrece los siguientes elementos para la creación de aplicaciones Web.

1.- Formularios Web

Los formularios Web representan las páginas que conforman una aplicación Web, y permiten la interacción con el usuario solicitando y mostrando información. Estos sirven de contenedores para los diferentes controles que conforman el aspecto y funcionalidad de una página ASP.NET.

2.- Controles Web y Controles HTML

Los controles Web y HTML conforman el contenido de las páginas de una aplicación Web permitiendo mostrar e introducir información.

Los controles Web poseen eventos, métodos y propiedades manipulables mediante código de servidor. Estos controles no son interpretables por los navegadores y se convierten en controles HTML que son enviados al cliente. Estos controles reciben también el nombre de “*controles del lado del servidor*”.

Los controles HTML se componen de código HTML y Javascript que es interpretado por el navegador del usuario. Estos controles reciben el nombre de “*controles del lado del cliente*”.

3.- Objetos ASP.

Son objetos gestionados por la propia plataforma ASP.NET, que están disponibles en todas las aplicaciones a través de propiedades de la clase *Page*. Estos objetos permiten diferentes operaciones:

- **Propiedad Request:** (objeto de la clase *HttpRequest*): Representa la solicitud de una página desde un cliente. Este objeto permite recuperar información de la petición y del cliente tales como cookies existentes, o datos enviados desde un formulario.
- **Propiedad Response:** (objeto de la clase *HttpResponse*): Representa el flujo de datos que conforman la respuesta enviada al navegador del cliente solicitante. Puede emplearse para añadir información en la respuesta al cliente o insertar cookies.
- **Propiedad Session:** (objeto de la clase *HttpSessionState*) Representa una colección de datos asociados exclusivamente a un usuario de la aplicación Web.
- **Propiedad Application:** (objeto de la clase *HttpApplicationState*) Representa una colección de datos asociados a una aplicación Web accesibles desde cualquier sesión.
- **Propiedad Server:** (objeto de la clase *HttpServerUtility*) Representa el propio servidor Web permitiendo ciertas operaciones como el redireccionamiento de peticiones, resolución de direcciones URL.. etc.
- **Propiedad Context:** (objeto de la clase *HttpContext*) Representa la información de contexto de una página donde se almacenan los datos que han de preservarse en su recarga. (*postback*).

Codificación en ASP.NET

3.1.- OPCIONES DE CODIFICACION EN ASP.NET

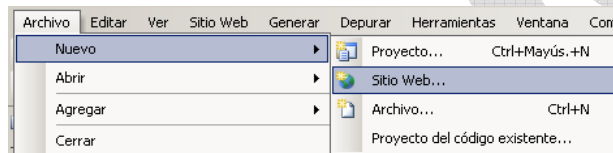
En ASP.NET existen dos formas de organizar el diseño y el código de las páginas que componen una aplicación Web.

3.1.1.- Código en línea

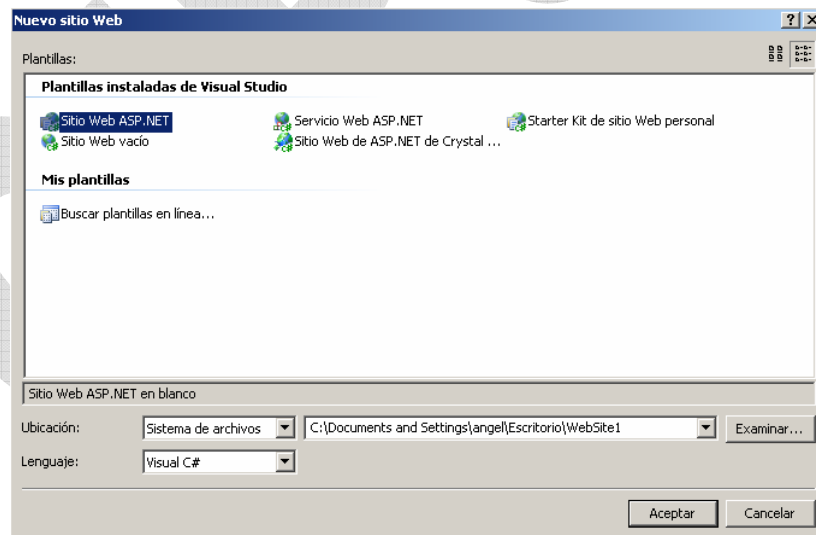
Consiste en incluir el código de servidor dentro del código HTML de la página. Este es el modo de codificación empleado en la antigua versión de ASP. Ejemplo:

Abrir el Visual Studio 2005 y crear un nuevo proyecto ASP.NET:

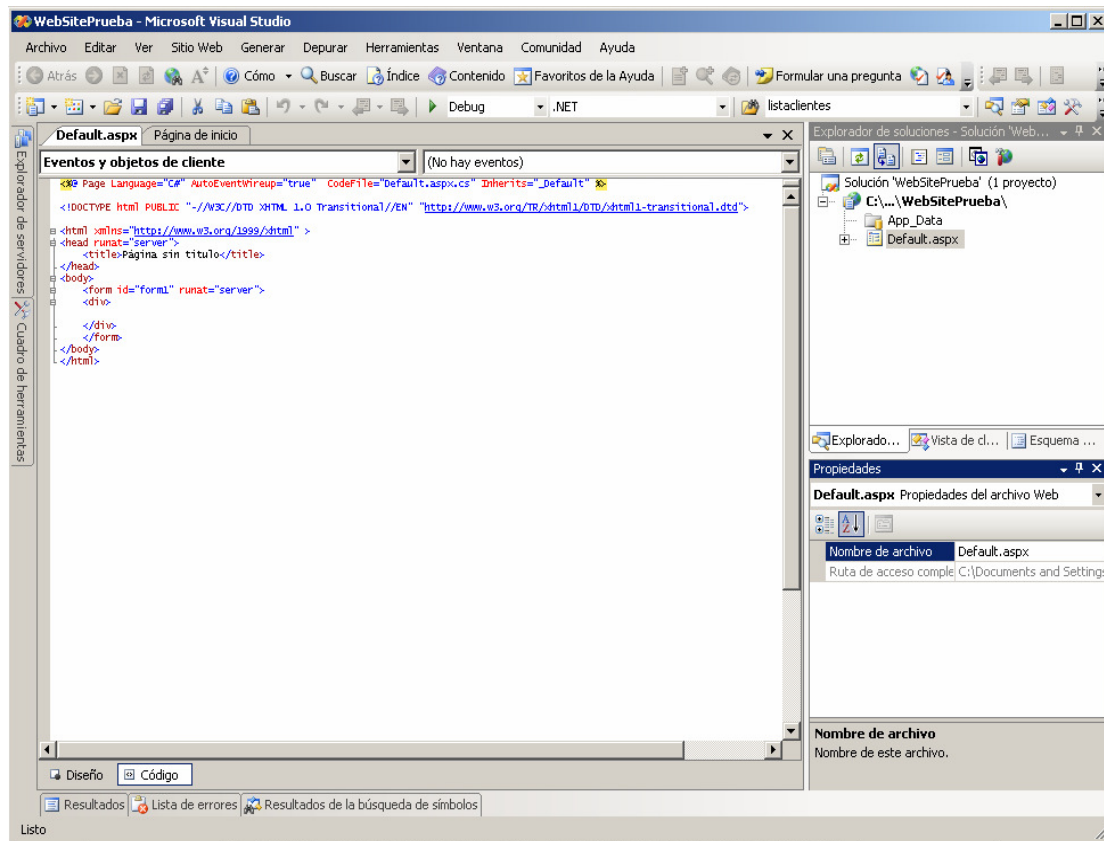
Archivo → Nuevo → Sitio Web



En la ventana “Nuevo sitio Web” seleccionaremos la opción “Sitio Web ASP.NET”, la ubicación donde queremos que se almacene el sitio Web, y el lenguaje de codificación: C#.



El IDE se abrirá mostrando el código del diseño HTML inicial de la página principal de la aplicación que acabamos de crear:



La ventana la vista de código de la página principal (default.aspx) muestra el código HTML asociado a la página. Esta se denomina también *vista de diseño*.

Se pueden emplean tres etiquetas distintas para incrustar código de servidor en el código HTML de la página.

→ **ETIQUETAS <%..%>**: Encerrado dentro de los delimitadores <%%> se encuentra todas las sentencias del código de servidor. Por ejemplo; el comando <%nombre="Pepe"%> asigna el valor Pepe a la variable nombre. Entre los delimitadores <%%> se pueden incluir varias sentencias en distintas líneas de código.

```
<%DateTime ahora=DateTime.Now;%>
```

Dentro de los delimitadores de código de servidor se pueden encontrar también instrucciones del lenguaje de correspondiente, así por ejemplo puede aparecer una instrucción if...else del lenguaje C#:

```
<%String nombre="", variable="";
if(nombre==""){
    variable="Nombre desconocido";
}else{
    variable="Hola amigo "+nombre;
}%>
<font color="green"><%=variable%></font>
```

También se puede incluir código HTML entre las instrucciones del lenguaje de servidor, aunque no es recomendable de cara a una lectura del código más sencilla.

```
<font color="green">
<%String nombre="";
if(nombre=="") {%>
Nombre desconocido
<%} else {%>
Hola amigo <%=nombre%>
<%}%>
</font>
```

También para poder realizar una lectura más sencilla del código ASP .NET se recomienda utilizar los delimitadores del código de servidor encerrando varias líneas de código en lugar de un par de etiquetas por cada línea.

```
<%String strNombre="";%>
<%Session.Add("nombre","Angel");%>
<%strNombre=Session["nombre"].ToString();%>
<%Response.Write(strNombre);%>
```

→ ETIQUETAS <%= %>: Dentro del segundo tipo de delimitadores <%= %> se encuentran expresiones que devuelve algún valor para ser mostrado en la salida de la página, así por ejemplo la expresión <%=nombre%> enviará al navegador el valor Pepe, es decir, el valor actual de la variable. Los delimitadores <%= %> sólo podemos encerrar una sentencia por línea.

```
<%=ahora.ToString()%>
```

→ ETIQUETAS <script>..</script>: El tercer tipo de delimitador es una etiqueta de la siguiente forma <script language="C#" runat="server"></script>. Esta etiqueta es utilizada únicamente para declarar métodos (procedimientos y funciones) de la página, y el delimitador <%%> sólo para escribir código que se ejecute dentro de la página fuera de procedimientos y funciones, es decir, El código que se ejecuta en línea.

```
<script language="c#" runat="server">
void Pulsado(Object sender, EventArgs args){
    etiqueta.Text="¡Hola Mundo!";
}
</script>
```

Supongamos que deseamos que la página muestre la fecha actual del servidor Web. Para ello, podemos incrustar el código C# que devuelve la fecha actual del sistema directamente en el HTML de la página, empleando las etiquetas de código en línea comentadas anteriormente. El código puede incluirse como una expresión que devuelve un valor el cual es inscrito en el código HTML al ejecutarse:

```
<%=DateTime.Now.ToShortDateString()%> // Expresion
```

Y también puede incluirse como un comando que ordena la inscripción del expresión en el código HTML al ejecutarse:

```
<% Response.Write(DateTime.Now.ToShortDateString()); %> <br /> // Comando
```

En ambos casos, el código C# inscrito en el HTML es ejecutado al ir a enviarse la página al usuario. El valor de la expresión, o el resultado del comando; son entonces inscritos en el HTML sustituyendo el código C# original, y la página es enviada al usuario como queda.

El código de la página quedaría así:

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Página sin título</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <%=DateTime.Now.ToShortDateString()%>
    </div>
  </form>
</body>
</html>
```

Para ejecutar la aplicación Web y ver el resultado seleccionamos primero la opción de compilación:

Generar → Generar Sitio web.

Si todo está correcto; se nos mostrará en la ventana de resultados el siguiente texto:

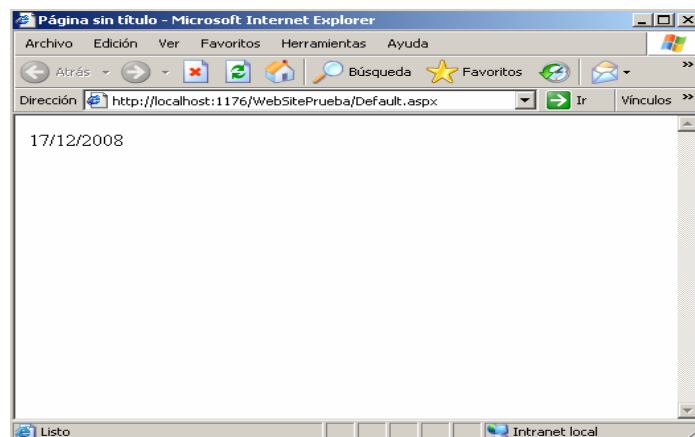
```
Validación completada
===== Generar: 1 correctos o actualizados, 0 incorrectos, 0 omitidos =====
```

A continuación lanzamos la aplicación Web, de igual manera que si un usuario solicitase la página principal (default.aspx) al servidor Web. Para ello seleccionamos la opción de inicio de depuración:

Depurar → Iniciar depuración.

*** Recordar que para esta operación el Visual Studio solicita permiso para modificar la configuración de la aplicación para que se ejecute en modo de depuración mostrando información de los errores que puedan surgir. Es conveniente permitirlo.**

Nos aparecerá la siguiente pantalla:



Como ejemplo obsérvense las diferencias entre el código HTML de la aplicación Web, y el que obtiene el navegador del cliente al solicitar la página.

Código página ASPX en el servidor Web

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Página sin título</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <%=DateTime.Now.ToShortDateString()%>
    </div>
  </form>
</body>
</html>
```

Código HTML que recibe el usuario Web

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
  Página sin título
</title></head>
<body>
  <form name="form1" method="post" action="Default.aspx" id="form1">
    <div>
      <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJOTU4MjMyMzI1ZGQSezRlqsIWWjDLOjoSNUOQFYfh+w==" />
    </div>
    <div>
      17/12/2008
    </div>
  </form>
</body>
</html>
```

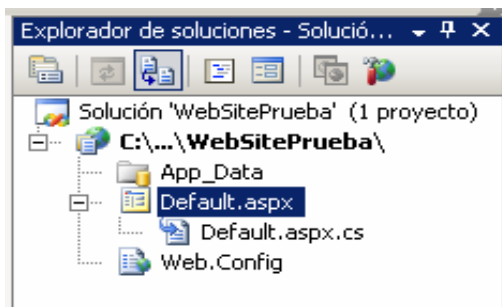
Como puede verse, el código en línea existente en el HTML desaparece siendo sustituido por su resultado tras la ejecución en el servidor antes de enviar la página al usuario Web.

3.1.2.- Código en el lado

Consiste en incluir el código de servidor en un fichero aparte. De esta manera, por cada página se crean dos ficheros: el fichero de diseño con extensión .aspx, y el fichero de código con extensión .aspx.cs si se utiliza C# como lenguaje para el código de servidor o .aspx.vb si se emplea Visual Basic.

Este es el modo de codificación preferido para ASP.NET ya que permite separar el interfaz o diseño de la página de la lógica o implementación de la misma.

Siguiendo con el ejemplo anterior:



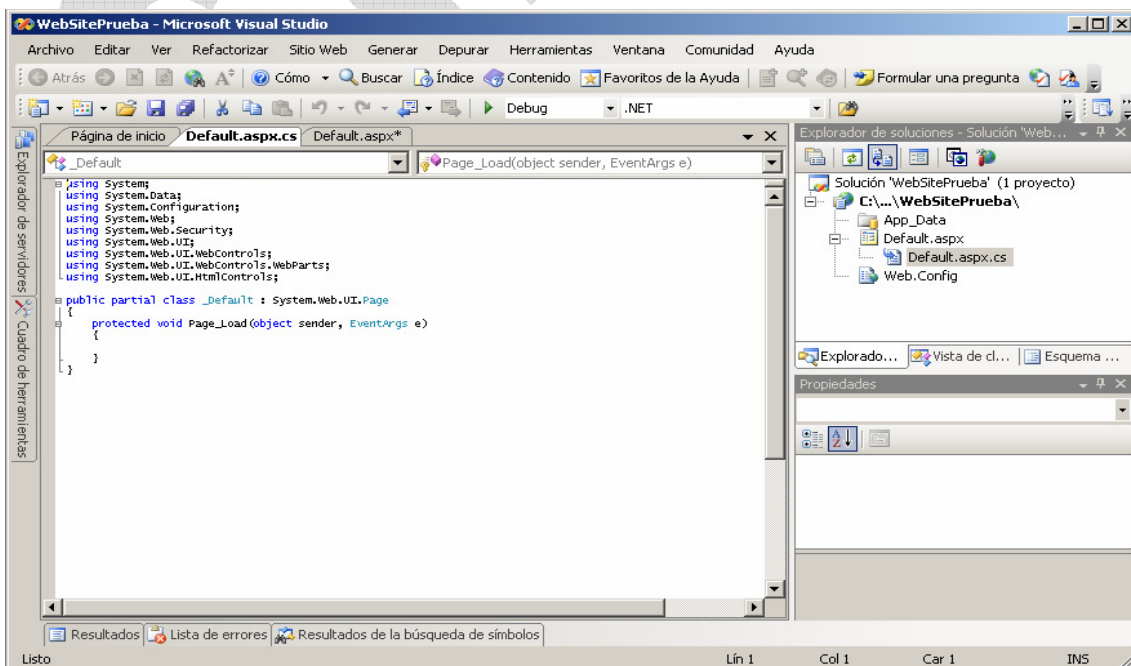
El fichero Default.aspx representa la página inicial de la aplicación Web.

El fichero Default.aspx.cs representa el fichero de código asociado a la página. Este fichero es el que contiene el código C# asociado a los eventos de la página y los controles contenidos en la misma.

La relación entre ambos ficheros (el fichero de página, y el de código), se establece mediante el atributo *CodeFile* de la directiva *Page* que aparece en la primera línea del fichero de la página Default.aspx:

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

Si seleccionamos el fichero de código Default.aspx.cs, se nos mostrará el contenido del fichero de código asociado a la página:



El código muestra la declaración parcial de una clase llamada `_Default`. Esta clase representa a la propia página Web y se hereda de la clase `System.Web.UI.Page`, clase padre para todas las páginas Web en ASP.NET.

La clase define un conjunto de métodos que se ejecutan como respuesta a eventos que se producen en la propia página o en los componentes que contiene. Por defecto el primer método que se muestra es `Page_Load`. Este método encapsula el código del evento Load de la página, el cual se produce al cargarse en memoria la misma tras ser solicitada.

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

Para hacer que la página muestre la fecha del servidor puede situarse el código correspondiente dentro del método `Page_Load` para se ejecute justo al cargarse la página tras recibirse su solicitud en el servidor Web.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
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.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write(DateTime.Now.ToShortDateString());
    }
}
```

El método `Write` del objeto `Response` hace que el texto pasado como parámetro sea inscrito al principio del código HTML enviado como respuesta al navegador del usuario:

```
17/12/2008
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
    Página sin título
</title></head>
<body>
    <form name="form1" method="post" action="Default.aspx" id="form1">
    <div>
    <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJNZgzNDMwNTMzZGS8aeBwkj19fnknH9iw1Y198nVCNw==" />
    </div>
    <div>
    </div>
    </form>
</body>
</html>
```


CLIPSA

La clase Page

La clase *Page* (*System.Web.UI.Page*) representa una página Web genérica sin ningún contenido. Cada página de una aplicación Web se crea en base a una clase que hereda a su vez de la clase *Page*. Cada objeto *Page* posee múltiples propiedades y comportamientos que permiten su funcionamiento heredados de *Page*. Entre las propiedades pueden distinguirse las siguientes como las más básicas:

- **IsPostBack:** Esta propiedad retorna un valor lógico falso si es la primera vez que se solicita la página, y cierto si se trata de una recarga de la misma. Se emplea en el evento *Load* de la página para controlar que sólo se ejecute la inicialización de la página la primera vez que se solicita.
- **EnableViewState:** Propiedad que habilita o deshabilita el mantenimiento del estado de los controles en la página durante las recargas de la misma.
- **Session:** Propiedad que permite acceder a una colección de pares clave-valor que permite almacenar información asociada a cada usuario de la aplicación en un determinado momento. Esta información recibe el nombre de *Sesión del usuario*. Se crea una sesión para cada usuario en el momento que solicita por primera vez una página cualquiera. La sesión permanece hasta que el usuario cierra el navegador, o transcurre un tiempo recibirse ninguna solicitud por su parte.
- **Application:** Propiedad que permite acceder a una colección pares clave-valor que permite almacenar información accesible para todos los usuarios de la aplicación en un determinado momento.
- **Request:** Propiedad que instancia un objeto de la clase *HttpRequest* que contiene información referente a la petición recibida desde el navegador del usuario. Permite obtener información acerca del navegador empleado por el usuario y su configuración
- **Response:** Propiedad que referencia un objeto de la clase *HttpResponse* que contiene el código generado por la página ASP.NET a enviar al navegador del usuario como respuesta a la petición. Puede emplearse para enviar cookies al navegador del cliente, o redireccionarlo a otra página.
- **Server:** Propiedad que referencia un objeto de la clase *HttpServerUtility* que permite realizar ciertas operaciones como convertir cadenas en código HTML válido.
- **User:** Propiedad que obtiene la información de usuarios autenticados.

Cada página Web puede a su vez contener diferentes controles de servidor de igual manera que un formulario en una aplicación de Windows contiene controles. Los controles ocupan un espacio en el código del diseño de la página y sus propiedades pueden fijarse en el diseño y modificarse desde el código. Como ejemplo; las siguientes líneas muestran la declaración de una caja de texto en una página ASP.NET.

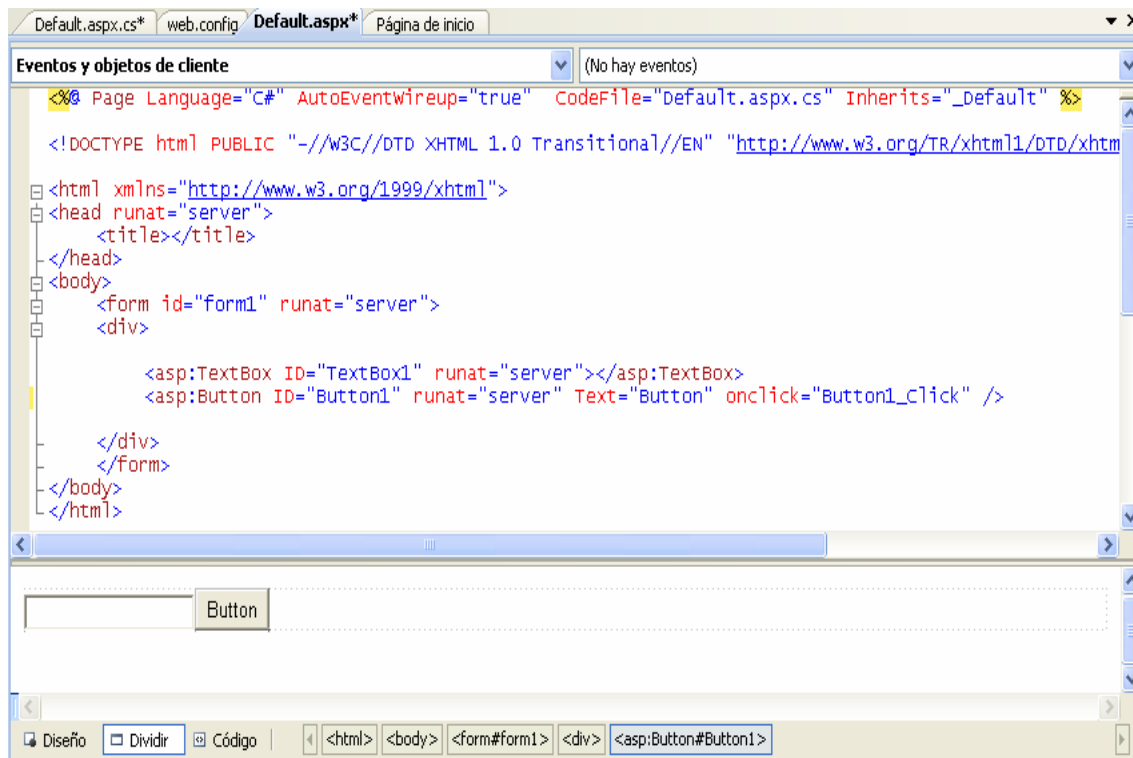
código .aspx: HTML (declaración del control en el diseño)

```
<asp:TextBox ID="TextBox1" Text="HOLA MUNDO" runat="Server"></asp:TextBox>
```

código .cs: C# (manipulación del control durante ejecución)

```
this.TextBox1.Text = "HOLA PAGINA";
```

Supóngase la siguiente página Web ASP.NET provista de dos controles Web; una caja de texto (control *TextBox*), y una botón (control *Button*). Los controles Web se incluyen en el diseño de la página dentro del código HTML de la misma:



Los controles Web disponen de eventos que permiten asociarles comportamientos antes ciertas acciones del usuario o sucesos. Las acciones asociadas a estos eventos se implementan en el fichero .cs dando lugar a la lógica o comportamiento de la página.

Por ejemplo; si queremos que la caja de texto muestre “TextBox1” muestre el mensaje “HOLA MUNDO” cuando el usuario pulse el botón “Button1”. Para conseguirlo debe implementarse el evento “Click” del botón “Button1”:

```
















public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        this.TextBox1.Text = "HOLA MUNDO";
    }
}
    
```

4.1.- EVENTOS DE PAGINAS ASP.NET

Además de los eventos asociados a los controles Web existen también otros eventos que tienen como origen la propia página.

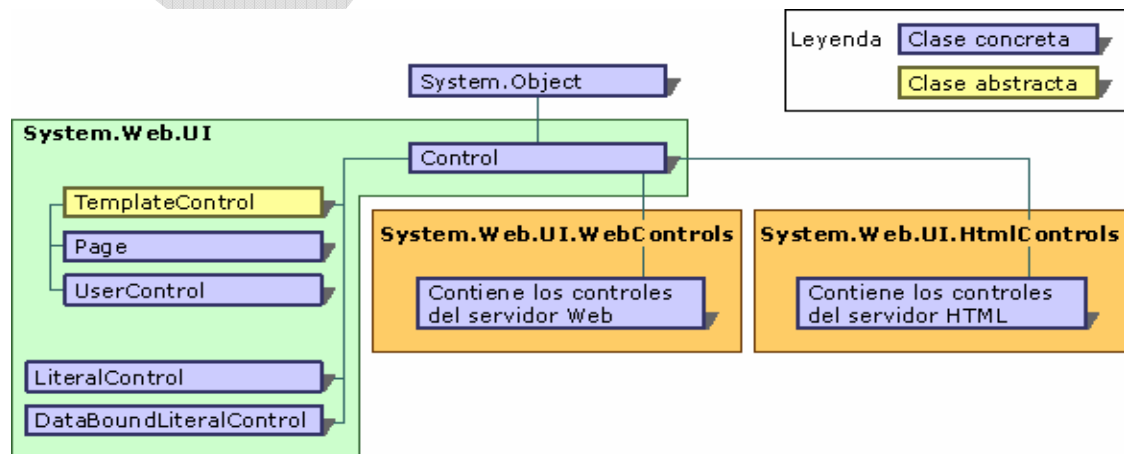
En la siguiente tabla se muestran la totalidad de los eventos de la clase Page.

Nombre	Descripción
 AbortTransaction	Se produce cuando un usuario finaliza una transacción. (Se hereda de TemplateControl).
 CommitTransaction	Se produce cuando finaliza una transacción. (Se hereda de TemplateControl).
 DataBinding	Se produce cuando el control de servidor se enlaza a un origen de datos. (Se hereda de Control).
 Disposed	Se produce cuando un control de servidor se libera de la memoria, lo que constituye la última fase del período de duración de un control de servidor cuando se solicita una página ASP.NET. (Se hereda de Control).
 Error	Se produce cuando se produce una excepción no controlada. (Se hereda de TemplateControl).
 Init	Tiene lugar al inicializar el control de servidor, que es el primer paso en su ciclo de vida. (Se hereda de Control).
 InitComplete	Se produce cuando se completa la inicialización de la página.
 Load	Se produce cuando el control de servidor se carga en el objeto Page . (Se hereda de Control).
 LoadComplete	Se produce al final de la fase de carga del ciclo de vida de la página.
 PreInit	Se produce al principio de la inicialización de la página.
 PreLoad	Se produce antes del evento Load de la página.
 PreRender	Se produce una vez que se carga el objeto Control , pero antes de su representación. (Se hereda de Control).
 PreRenderComplete	Se produce antes de que se represente el contenido de la página.
 SaveStateComplete	Se produce después de que la página ha terminado de guardar toda la información de estado de vista y estado de control para la página y los controles de la página.
 Unload	Se produce cuando el control de servidor se descarga de la memoria. (Se hereda de Control).

Algunos de estos eventos son específicos de la clase Page, mientras que otros se heredan de una clase superior llamada System.Web.UI.Control.

La clase Control define una serie de eventos propios que son heredados por la clase Page: Init, Load, Prerender, Unload, Dispose y DataBinding. Estos eventos también pueden encontrarse en los controles de servidor debido a que también heredan de la clase Control al igual que Page.

El siguiente esquema muestra las relaciones de herencia de la clase Control con la clase Page y las clases que conforman los controles de servidor.



Es importante capturar los eventos de página para poder ligar la ejecución de ciertas operaciones a ellos. Existen dos maneras de capturar estos eventos:

1.- El uso de métodos delegados: Un delegado es un método que se invoca automáticamente al producirse un determinado evento. Estos métodos permiten encapsular un código para que se ejecute cuando suceda el evento correspondiente:

```
public partial class Default2 : System.Web.UI.Page
{
    protected void Page_PreInit(object sender, EventArgs e)
    {
    }
    protected void Page_Init(object sender, EventArgs e)
    {
    }
    protected void Page_PreLoad(object sender, EventArgs e)
    {
    }
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Page_PreRender(object sender, EventArgs e)
    {
    }
    protected void Page_Unload(object sender, EventArgs e)
    {
    }
}
```

Estos métodos se asocian de por sí al evento correspondiente de la clase Page, siempre que la propiedad AutoEventWiredUp esta fijada a true:

```
<%@ Page language="c#" Trace="false" CodeFile="Test.aspx.cs" AutoEventWireup="true"
Inherits="Test.Input" %>
```

2.- Sobrescribir los métodos heredados de la clase Page encargados de provocar los distintos eventos.

```
public partial class Default2 : System.Web.UI.Page
{
    protected override void OnPreInit(EventArgs e)
    {
        base.OnPreInit(e);
    }
    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);
    }
    protected override void OnPreLoad(EventArgs e)
    {
        base.OnPreLoad(e);
    }
    protected override void OnLoad(EventArgs e)
    {
        base.OnLoad(e);
    }
    protected override void OnPreRender(EventArgs e)
    {
        base.OnPreRender(e);
    }
    protected override void OnUnload(EventArgs e)
    {
        base.OnUnload(e);
    }
}
```

En el caso de emplear la sobrescritura de los métodos encargados de provocar los eventos, es importante respetar siempre la invocación al método de la clase padre para evitar problemas en el procesamiento de la página.

En el momento de ejecutarse la llamada al método correspondiente de la clase Padre, se produce el evento que dispararía el método delegado si es que estuviese declarado también.

Ejemplo: Sea la siguiente sección de código:

```
protected override void OnLoad(EventArgs e)
{
    Response.Write("Antes del base.OnLoad");
    base.OnLoad(e);
    Response.Write("Despues de base.OnLoad");
}
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("Delegado evento Load.");
}
```

Al ejecutarse la página; el código generado para el usuario sería el siguiente:

```
Antes del base.OnLoad
Delegado evento Load.
Después de base.OnLoad.
```

4.2.- EVENTOS DE APLICACIÓN

A parte de los eventos propios de las páginas y los controles de servidor que contienen; existen otros eventos que afectan a la propia aplicación.

Los eventos de aplicación responden sucesos globales que afectan a la aplicación independientemente de la página que el usuario esté utilizando. Los principales eventos de aplicación son los siguientes:

Application_Start: Evento que se produce cuando la aplicación Web se inicia, esto es; cuando recibe la primera solicitud de cualquier usuario tras el inicio o reinicio del servidor Web.

Application_End: Evento que se produce cuando la aplicación Web es apagada al reiniciar o apagar el servidor Web.

Application_BeginRequest: Evento que se produce cada vez que se recibe la solicitud de una página desde el navegador de un usuario. El evento se produce justo antes de procesarse la página solicitada.

Application_EndRequest: Evento que se produce cada vez que se solicita una página de la aplicación Web justo después de procesarse la misma.

Session_Start: Evento que se produce cuando se inicia una nueva sesión al solicita un usuario una página por primera vez tras iniciar su navegador.

Session_End: Evento que se produce al cerrarse la sesión.

Application_Error: Evento que se produce cuando una página provoca una excepción que no resulta capturada.

Los eventos de aplicación se implementan en un archivo de código específico denominado *global.asax*. Para añadir este archivo a una aplicación Web ASP.NET en desarrollo con Visual Studio debe seleccionarse la opción:

Sitio Web → Agregar nuevo elemento... → Clase de aplicación global.

El código generado por defecto para el archivo global.asax incluye métodos para atender los siguientes eventos:

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Start(object sender, EventArgs e) {
        // Código que se ejecuta al iniciarse la aplicación
    }

    void Application_End(object sender, EventArgs e) {
        // Código que se ejecuta cuando se cierra la aplicación
    }

    void Application_Error(object sender, EventArgs e) {
        // Código que se ejecuta al producirse un error no controlado
    }

    void Session_Start(object sender, EventArgs e) {
        // Código que se ejecuta cuando se inicia una nueva sesión
    }

    void Session_End(object sender, EventArgs e) {
        // Código que se ejecuta cuando finaliza una sesión.
        // Nota: El evento Session_End se desencadena sólo con el modo sessionstate
        // se establece como InProc en el archivo web.config. Si el modo de sesión se
        // establece como StateServer
        // o SQLServer, el evento no se genera.
    }
}
</script>
```

4.3.- CICLO DE VIDA DE PAGINAS ASP.NET

Cada vez que una página es solicitada al servidor Web, se producen una serie de operaciones desde su carga en memoria hasta su descarga una vez enviada la respuesta al usuario. La sucesión de estos estados produce ciertos eventos que pueden aprovecharse para llevar a cabo determinadas acciones:

4.3.1.- Evento de Inicialización

La página y los controles que contiene son cargados en memoria al recibirse la solicitud de la misma. El evento de inicialización puede ser tratado mediante el método delegado *Page_Init*, o sobrescribiendo el método *OnInit*.

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

protected override void OnInit(EventArgs e)
{
    base.OnInit(e);
}
```

En este punto se establecen las propiedades *Request*, y *Response* que proporcionan acceso a los datos de la petición, y la respuesta enviada al usuario. También se establece el valor booleano de la propiedad *isPostBack* que determina si la página se solicita por primera vez, o se trata de una recarga.

4.3.2.- Carga del estado de página (*viewstate*)

Se recupera el estado de página o *viewstate* asociado a la página y sus controles; que consiste en un conjunto de pares de valores clave-valor que ASP.NET gestiona para permitir el paso de información de una página a la siguiente. Esta operación es llevada a cabo mediante el método de la clase *LoadViewState* heredado de la clase *Page*.

```
protected override void LoadViewState(object o)
```

Los controles de la página recuperan entonces el estado que tenían en el lado del cliente al solicitar la recarga. Para ello, sus estados son enviados por el navegador junto con la solicitud de recarga empleando campos ocultos de HTML.

Es importante destacar que las páginas Web no mantienen por si mismas el estado de los controles que contienen. Cada vez que se produce una solicitud de una página, en el servidor se crea una nueva instancia para atenderla y ser eliminada inmediatamente después de enviar la respuesta. Debido a ello, cuando se produce una nueva petición de la página (*postback*), la nueva instancia que se crea no tiene los datos manejados por la instancia de la petición anterior. El estado de página permite enviar de vuelta al servidor el estado de los controles de la página, de modo que éstos se mantengan entre peticiones de la misma página.

4.3.3.- Evento de Carga.

Se inicializan las instancias de los objetos que representan todos los controles presentes en la página, permitiendo el acceso a todas sus propiedades y métodos mediante código. El evento puede ser tratado mediante el método delegado *Page_Load*, o sobrescribiendo el método *OnLoad*.

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

protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);
}
```

En este estado, los valores de las propiedades de los controles ya son disponibles para su manipulación. Debido a ello suele ser el evento más utilizado comúnmente para implementa cualquier tipo de acción inicial al solicitarse la página.

4.3.4.- Gestión de Eventos

Una vez cargados todos los controles se comprueba si sus estados han cambiados desde la anterior solicitud de recarga. En caso de detectarse un cambio en el estado de un control se invoca el método *RaisePostDataChanged* del control. Este método es el encargado de activar el evento del control que corresponda.

Por ejemplo: En el caso del control Web TextBox, el método *RaisePostDataChanged* provoca la llamada al método *OnTextChanged* que provoca el evento *TextChanged*. Este evento señala un cambio en el contenido de la caja de texto por parte del cliente sin provocar una recarga de la página.

Una vez atendidos todos los eventos asociados a cambios en el estado de los controles Web, se ejecuta entonces el método *RaisePostBackEvent* que invoca al evento correspondiente del control que provocó la solicitud de recarga de la página.

Por ejemplo: En el caso del control Web Button, la pulsación del mismo provoca una recarga automática de la página y la ejecución del evento Click del control al recibirse la solicitud en el servidor Web.

4.3.5.- Evento de PreRenderizado.

Este se produce justo antes de iniciar la generación de código HTML de respuesta para el navegador del usuario. El evento de pre-renderizado constituye la última ocasión para modificar por código el estado de un control de forma persistente, antes de que sea generada la respuesta HTML que se enviará al usuario.

El evento puede ser tratado mediante el método delegado *Page_PreRender*, o bien sobrescribiendo el método *OnPreRender*.

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

protected override void OnPreRender(EventArgs e)
{
    base.OnPreRender(e);
}
```

4.3.6.- Guardado del estado de página (*viewstate*)

El estado de página y sus controles (*viewstate*) es guardado definitivamente una vez realizadas todas las operaciones de atención a eventos. La información se envía al cliente codificada dentro de campos ocultos inscritos en el código HTML de la respuesta. Esta operación produce el evento *SaveStateComplete* que puede ser tratado empleando el método delegado *Page_SaveStateComplete*, o sobrescribiendo el método heredado *OnSaveStateComplete*.

```
protected void Page_SaveStateComplete(Object sender, EventArgs e)
{
}

protected override void OnSaveStateComplete(EventArgs e)
{
    base.OnSaveStateComplete(e);
}
```

4.3.7.- Renderización de la página.

En la renderización de la página, cada control genera su código HTML, que es inscrito junto con el código del resto de controles empleando una instancia de la clase *HtmlTextWriter*. Dicha instancia constituye un flujo de datos en el que se va añadiendo código HTML según se procesa la página con el fin de enviarlo al navegador del usuario.

4.3.8.- Evento de Descarga

El evento de descarga se produce en el momento de eliminar todos los objetos relacionados con la página. Se produce entonces el evento *Unload*, que puede atenderse empleando el método delegado *Page_Unload*, o sobrescribiendo el método origen *OnUnload*.

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

protected override void OnUnload(EventArgs e)
{
    base.OnUnload(e);
}
```

Debe tenerse en cuenta que las propiedades como *Request* y *Response* ya no se encuentran disponibles llegado este estado, y por lo tanto resulta imposible enviar ningún mensaje al usuario.

La siguiente tabla muestra esquemáticamente la secuencia y significado de cada uno de los eventos asociados a la ejecución de una página ASP.NET, desde el momento de su solicitud, hasta el momento de su eliminación de la memoria del servidor tras enviar el código HTML de respuesta para el navegador del cliente.

ESTADO	SIGNIFICADO
<i>Solicitud</i>	La solicitud de página se produce cuando el servidor Web recibe una petición de esa página. Se crean los objetos <i>Request</i> para capturar datos de la petición, y el objeto <i>Response</i> para enviar datos. Se inicializa el valor de la propiedad <i>IsPostBack</i> , que indica mediante un valor de tipo booleano si la página es solicitada por primera vez, o si se trata de una recarga.
<i>Inicialización</i>	Se inicializan los controles contenidos en la página, pero sin recuperar su valor anterior en caso de tratarse de una recarga. Se producen los eventos Init de los controles, y después el de la página atendido por el método <i>Page_Init</i> .
<i>Carga</i>	Se produce el evento Load atendido por el método <i>Page_Load</i> . Después se ejecuta produce el evento Load de cada uno de los controles contenidos en la página. Si se trata de una recarga, los controles contenidos en la página recuperan su estado anterior.
<i>Validación</i>	Se ejecutan los controles de validación
<i>Gestión Eventos</i>	Si se trata de una recarga, se ejecutan los métodos de respuesta a los eventos producidos en los controles Web
<i>Representación</i>	Se genera el código HTML que se envía como respuesta al usuario de la aplicación Web. Primero se produce el evento de la página atendido por el método <i>Page_PreRender</i> , seguido del de los controles Web.
<i>Descarga</i>	Enviado el código HTML de respuesta al usuario, la instancia en memoria de la página es destruida. Los objetos Response y Request ya han sido eliminados y no pueden invocarse. Se produce el evento Unload atendido por el método <i>Page_Unload</i> .

4.4.- EJECUCION DE PAGINAS ASP.NET

Cuando un cliente solicita una página ASP.NET de una aplicación Web tiene lugar en el servidor dos procesos. Por un lado se produce la compilación y generación de un ensamblado con el código de servidor asociado a la página. Por otro lado, se ejecuta el código del ensamblado y se genera el código HTML de respuesta para el navegador del usuario.

Las fases de este proceso son tres:

4.4.1.- Generación de clase parcial complementaria:

Si nos fijamos en el código asociado a cualquier página ASP, vemos:

```
public partial class _Default : System.Web.UI.Page
{
    ...
}
```

Es decir, cada página está definida por una clase (por defecto llamada `_Default`), que hereda a su vez de la clase `Page`. Se trata de una clase *parcial*; lo que indica que la declaración de esta clase no está completa. Pero, ¿ Qué le falta?.

Supóngase la siguiente página provista de una

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        this.TextBox1.Text = "HOLA MUNDO";
    }
}
```

En el código aparece una variable de instancia `TextBox1`, y un método `Button1_Click` supuestamente ligado al evento `Clic` del control `Button1`. Sin embargo; en ninguna parte del código aparece la declaración e instanciación del control representado por la instancia `TextBox1`.

Cuando se solicita la página, se genera un archivo de código que contiene precisamente la declaración e inicialización de las instancias de todos los controles contenidos en la página. Ese código de une con el ya existente en el fichero de código de la página. Ej:

```
public partial class _Default : System.Web.UI.Page
{
    Protected Button Button1;
    Protected TextBox TextBox1;
}
```

4.4.2.- Generación de clase final

Tras la generación del código de inicialización se crea una nueva clase completa. Esta clase contiene todo el código necesario para la generación y gestión de los comportamientos de la página, y se crea a partir del código HTML contenido en el fichero de diseño de la página (.aspx). Esta nueva clase hereda de la clase parcial contenida en el fichero de código asociado a la página, y cuyo nombre aparece en el valor del atributo Inherits de la etiqueta <page>:

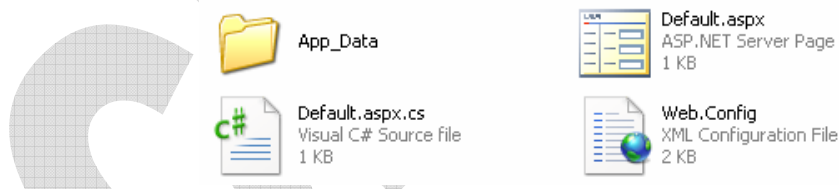
```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

Esta clase final reúne tanto la lógica de la aplicación con los métodos de respuesta a eventos, así como las instrucciones de generación del código HTML para las respuestas. Su código será el que se ejecute para atender todas las peticiones de la página.

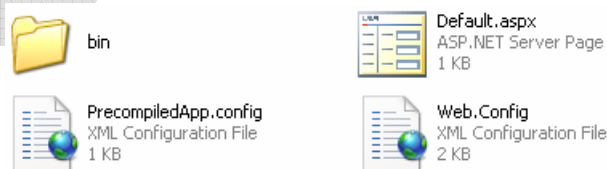
4.4.3.- Generación del ensamblado asociado a la página

La clase final es finalmente compilada y almacenada junto con la clase base de la que hereda en un ensamblado. Estos ensamblados son ficheros con extensión *dll* que se copian en un directorio temporal del servidor Web gestionado por el motor de ejecución de ASP.NET.

Ejemplo: A continuación se muestra la estructura básica de los ficheros que componen la aplicación Web que hemos empleado de ejemplo. Se aprecia el fichero de diseño de la página .Default.aspx, y el fichero de código asociado a la misma Default.aspx.cs.



Al ser solicitada la página por primera vez, se genera lo siguiente:



Aparentemente es lo mismo, pero ya no está el fichero de código Default.aspx.cs. En su lugar aparece una carpeta bin, que contiene:



El fichero contenido en la carpeta bin “App_Web_c-0dyqfe.dll” es el ensamblado producto de la compilación del código de la página Default.aspx.

Si abrimos el fichero Default.aspx, se aprecia una diferencia:

```
<%@ page language="C#" autoeventwireup="true" inherits="_Default, App_Web_c-0dyqfe" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Página sin título</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:TextBox ID="txtNombre" runat="server"></asp:TextBox>
    <asp:Button ID="btnAccion" runat="server" Text="Button"
onClick="btnAccion_Click" />
    <br />
    <asp:Label ID="lblEtiqueta" runat="server"></asp:Label>
  </form>
</body>
</html>
```

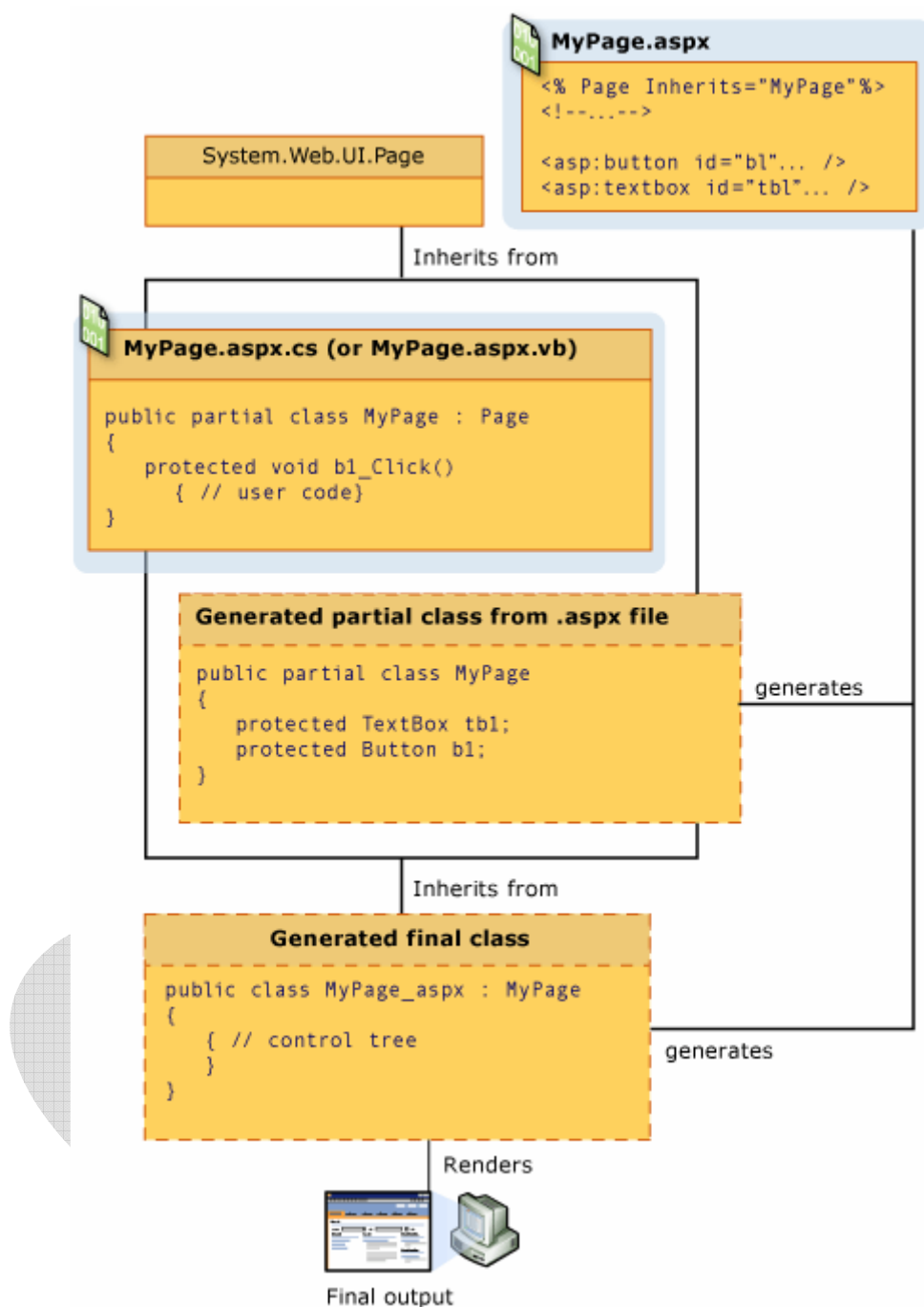
La etiqueta Page ya no contiene el atributo *CodeFile*, y el atributo *Inheris* ahora contiene el valor “_Default, App_Web_c-0dyqfe”. Estos cambios son debidos a que la página aspx, ya no está asociada a un fichero de código fuente en C#, si no a un fichero ensamblado de código compilado DLL.

Cada vez que se recibe una petición sobre una página, el motor de ejecución de ASP.NET crea una instancia de la clase asociada, ejecuta los métodos necesarios, y la elimina tras enviar el HTML de respuesta al usuario.

En el siguiente esquema se muestra la secuencia de sucesos entre el navegador del cliente y el servidor Web en ejemplo anterior:

CLIENTE	SERVIDOR
Petición Default.aspx	
	<ul style="list-style-type: none"> • Carga + Inicialización de la instancia de página • Generación HTML respuesta + Envío. • Eliminación de instancia
<ul style="list-style-type: none"> • Usuario escribe su nombre en la caja de texto txtNombre, y pulsa el botón btnAccion. • Petición de recarga de página. 	
	<ul style="list-style-type: none"> • Carga + Inicialización de la instancia de página • <i>Ejecución de método de respuesta btnAccion_Click</i> • Generación HTML respuesta + Envío. • Eliminación de instancia

La siguiente figura representa esquemáticamente las fases de ejecución de una página ASP.NET:



Objetos de Servidor ASP.NET

Los controles Web constituyen las piezas elementales con las que se diseñan las páginas en ASP.NET. La interacción con el usuario y el aspecto de las páginas depende del conjunto de controles Web que se utilicen. Sin embargo; el funcionamiento de una aplicación Web requiere de otros mecanismos a parte de la gestión de controles de página y el tratamiento de sus eventos.

ASP.NET dispone de una serie de clases auxiliares que permiten ciertas operaciones especiales necesarias para el funcionamiento de las aplicaciones Web:

- **HttpRequest:** Acceso a los datos de la petición por parte del usuario, así como información relativa al navegador del cliente.
- **HttpResponse:** Acceso a la secuencia de código HTML que se genera como respuesta para el usuario como respuesta a una petición.
- **HttpSessionState:** Control y mantenimiento de la información de cada usuario que está utilizando en un momento la aplicación Web.
- **HttpApplicationState:** Control y mantenimiento de información general de la aplicación Web empleada por todos los usuarios.
- **HttpServerUtility:** Control de las funciones del propio servidor de la aplicación Web.
- **HttpContext:** Acceso al contexto de ejecución de página para compartir mantener información entre páginas.

Todas estas clases pertenecen al espacio de nombres *System.Web*. Estas clases son instanciadas con cada petición y sus objetos son accesibles a través de ciertas propiedades de la clase Page.

Propiedad de System.Web.UI.Page	Clase
Request	HttpRequest
Response	HttpResponse
Session	HttpSessionState
Application	HttpApplicationState
Server	HttpServerUtility
Context	HttpContext

5.1.- EL OBJETO REQUEST (*HttpRequest*)

Recordemos que el protocolo que se emplea para la transferencia de páginas Web entre servidor y usuario es HTTP. Una petición HTTP además de solicitar una página guarda datos útiles del equipo del usuario, tales como los valores de los campos de un formulario, información de cookies... etc. Para poder recuperar esta información se emplea el objeto Request, que representa la propia petición del usuario.

En las páginas ASP.NET los propios controles de servidor se ocupan de mantener su estado entre solicitudes empleando los campos ocultos. Sin embargo; si una página ASP.NET es solicitada desde formularios HTML o Flash, en ese caso hay que recuperar manualmente la información de sus campos a través de la petición.

5.1.1.- Recuperación de datos de un formulario HTML

Los formularios en HTML tienen la única función de enviar los datos de sus campos a una página en una dirección URL indicada. Un formulario se identifica por la etiqueta *Form* que dispone de dos atributos básicos: *action* y *method*. El atributo *action* primero indica la URL de la página a la que se enviarán los datos del formulario. El segundo atributo *method* indica la forma en que se enviarán los datos.

Existen dos maneras de enviar la información de un formulario a la página indicada en el atributo *action* del formulario.

- **POST:** El conjunto de valores de los campos del formulario se envían codificados en el propio cuerpo de la petición HTTP al servidor. Este método se emplea para envío de información sensible que va a almacenarse, o a modificar la existente en una aplicación
- **GET:** El conjunto de valores de los campos del formulario se envían codificados en la propia dirección URL de la página solicitada. Este método se emplea para envío de información no delicada dedicada a hacer consultas o mostrar información (exceptuando operaciones de autenticación con contraseñas).

La principal diferencia entre ambos métodos radica en por donde se envían los datos. En el método POST, los valores de los controles de un formulario se guardan dentro de la propia petición HTTP que se envía al servidor. En el caso del método GET, los datos se envían concatenados con la URL de la página que se solicita: Ej:

<http://www.dominio.com/altas.aspx?nombre=jhonny&apellido=petrov&dni=29394482>

Evidentemente, esta técnica es mucho más insegura para datos delicados como contraseñas o modificaciones, ya que la URL puede falsearse con datos malintencionados sin dificultad.

5.1.2.- Recuperación de datos mediante POST (*Request.Form*)

Supongamos el siguiente formulario HTML convencional que envía la información de sus campos de texto y botones de opción a la página `altas.aspx`, empleando el método POST:

El formulario está compuesto por dos cajas de texto y dos botones de opción y un cuadro de lista de selección simple. El formulario debe incluir obligatoriamente un botón de tipo “submit” que ejecuta el envío de datos al ser pulsado.

Código HTML

```
<form action="altas.aspx" method="post">
  Nombre:
  <input id="txtNombre" name="campoNombre" type="text" /><br />
  Edad:
  <input id="txtEdad" name="campoEdad" type="text" /><br />
  Genero:
  <input id="rdoHombre" value="hombre" name="genero" type="radio" />Hombre
  <input id="rdoMujer" value="mujer" name="genero" type="radio" />Mujer
  <br />
  <input id="Submit1" title="Enviar datos" type="submit" value="Enviar
  Datos" />
  <select id="Select1" name="idioma" size="3">
    <option selected="selected" value="1">Castellano</option>
    <option value="2">Frances</option>
    <option value="3">Aleman</option>
    <option value="4">Ingles</option>
  </select>
</form>
```

Para recuperar el estado de cada uno de sus controles desde la página ASP.NET indicada en el atributo `action` de la etiqueta `Form`, debe emplearse la instancia de `HttpRequest` ligada a la página mediante la propiedad `Request`:

La clase `HttpRequest` posee una propiedad `Form` que contiene una instancia de la clase `NameValueCollection`. Esta instancia es una colección de pares clave-valor que contiene los nombres y valores de todos los controles en el formulario solicitante. Para acceder a un valor de un control determinado se puede emplear el valor de su atributo `name` como índice.

Por ejemplo, para recuperar el valor de los campos de texto del nombre y la edad del formulario, se emplearía el siguiente código:

```
// Recuperacion del nombre
String nombre = Request.Form["campoNombre"];
// Recuperacion de la edad
int edad = int.Parse(Request.Form["campoEdad"]);
```

Los botones de opción también se recuperan de la misma manera. En este caso, como el valor del atributo *name* es compartido por todos los botones de opción de un mismo grupo; el valor retornado se corresponde con el del atributo *value* del botón de opción seleccionado.

```
// Recuperacion del dato del genero
bool esMujer;
if (Request.Form["genero"].Equals("hombre"))
    esMujer = false;
else
    esMujer = true;
```

En el caso de la lista de elementos, el valor devuelto se corresponde con el del atributo *value* del elemento seleccionado. En el caso de permitirse la selección múltiple de elementos, el valor devuelto contiene los atributos *value* de los elementos seleccionados separados entre ellos por comas.

```
const int CASTELLANO = 1;
const int FRANCES = 2;
const int ALEMAN = 3;
const int INGLES = 4;
int idioma = int.Parse(Request.Form["idioma"]);
switch (idioma)
{
    case CASTELLANO:
        break;
    case FRANCES:
        break;
    case ALEMAN:
        break;
    case INGLES:
        break;
}
```

Si se desea ver todos los datos enviados por el formulario solo hay que recorrer los elementos de la colección Form:

```
protected void Page_Load(object sender, EventArgs e)
{
    int indx;
    System.Collections.Specialized.NameValueCollection coleccion;

    // obtencion del objeto coleccion de la propiedad Form de Request.
    coleccion=Request.Form;
    // obtencion de todos los nombres de los elementos en la coleccion
    String[] nombreControles = coleccion.AllKeys;
    Response.Write("<table>");
    Response.Write("<tr><th>CONTROL</th><th>VALOR</th></tr>");
    for (indx = 0; indx < nombreControles.Length; indx++)
    {
        Response.Write("<tr>");
        Response.Write("<td>" + nombreControles[indx] + "</td>");
        Response.Write("<td>" + coleccion[indx] + "</td>");
        Response.Write("</tr>");
    }
    Response.Write("</table>");
}
```

Ejemplo:

Microsoft Internet Explorer window titled 'Página sin título'. The address bar shows 'http://localhost:2806/SitioWeb/HTMLPage.htm'. The form contains the following fields:

- Mombre: Irina Ivanova
- Edad: 25
- Genero: ☐ Hombre ☒ Mujer
- Language dropdown: Frances, Aleman, Ingles (selected)
- Button: Enviar Datos

Página HTML formulario.

En el momento que pulsemos el botón Enviar Datos, el navegador lanzará una petición HTTP solicitando la página ASP.NET. Al ser una petición lanzada desde formulario, ésta lleva incluida la información de los controles presentes en el formulario responsable de la solicitud.

Microsoft Internet Explorer window titled 'Página sin título'. The address bar shows 'http://localhost:2806/SitioWeb/Default2.aspx'. The response is a table with the following data:

CONTROL	VALOR
campoNombre	Irina Ivanova
campoEdad	25
genero	mujer
idioma	4

Respuesta página ASP.NET mostrando la relación de nombres y valores de los campos del formulario

El código del evento Load de la página recorre entonces la colección de campos y valores presente en la petición mediante la propiedad Form del objeto Request. El resultado son los nombres de todos los controles del formulario solicitante con sus respectivos valores.

Es importante destacar que la operación de solicitud de una página ASP.NET por parte de un formulario HTML, no constituye una recarga de página. Sólo se considera recarga cuando se trata de una solicitud desde una instancia de la propia página para ejecutar en el servidor los eventos ligados a los controles Web presentes en la misma.

5.1.3.- Recuperación de datos mediante GET (*Request.QueryString*)

Otra forma de enviar información de una página a otra es insertarla en la propia URL de la Web a la que se invoca. Esto es común verlo en las páginas de motores de búsqueda tipo google.

<http://www.google.com/search?query=asp&lang=esp>

Los datos se concatenan con la URL de la página a la que queremos acceder mediante un signo de interrogación:

<url>?<clave=valor>&<clave=valor>...

Suponiendo el ejemplo de las dos páginas ASP.NET anteriores:

Usuario: Clave:

El código de diseño es idéntico pero eliminando la propiedad *PostBackUrl* del botón.

```
<body>
  <form id="form1" runat="server">
    <div>
      Usuario: <asp:TextBox ID="txtUser" runat="server"></asp:TextBox>
      Clave: <asp:TextBox ID="txtPassword" runat="server"></asp:TextBox>
      <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click"/>
    </div>
  </form>
</body>
```

A diferencia del caso anterior es necesario capturar el evento *Click* asociado al botón Web. El objetivo es provocar desde código el redireccionamiento a la página *clave.aspx* mediante una URL. Por otro lado; la URL debe ser modificada para agregar los valores a enviar a la página solicitada.

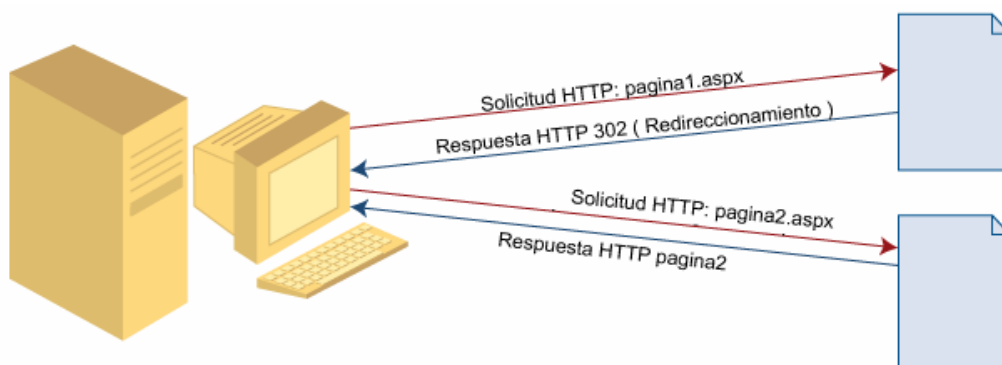
```
protected void Button1_Click(object sender, EventArgs e)
{
    String url = "datos.aspx";
    url += "?usuario=" + Server.UrlEncode(this.txtUser.Text);
    url += "&clave=" + Server.UrlEncode(this.txtPassword.Text);
    Response.Redirect(url);
}
```

Cada valor agregado a la URL debe ir separado por un signo & y acompañado por un campo clave que lo identifica. En este caso, los valores claves son "usuario" y "clave". Para recuperar los valores agregados en la URL cuando la solicitud llega al servidor debe emplearse la propiedad *QueryString* del objeto *Request*. Esta propiedad hace referencia a una colección que recogen los valores agregados en la URL como un conjunto de pares clave-valor.

De este modo, si introducimos como nombre *Roger* y como contraseña *12345*, la URL generada resultará:

`datos.aspx?usuario=Roger&clave=12345`

Por último, la invocación del método *Redirect* de la propiedad de la página *Response* fuerza el redireccionamiento a la página de la URL pasada como parámetro. Para ello se envía al navegador del usuario una respuesta HTTP302 que fuerza la solicitud a su vez de la página indicada en la URL.



Para obtener los valores debe emplearse la propiedad *QueryString* del objeto *Response*. Esta propiedad referencia a una colección de pares clave-valor que se carga automáticamente con los valores agregados en la URL. Para obtener cada dato almacenado en la colección *QueryString* debe indicarse la clave correspondiente a cada valor mediante el operador de indización [].

```
protected void Page_Load(object sender, EventArgs e)
{
    this.lblUsuario.Text = Server.UrlDecode(Request.QueryString["usuario"]);
    this.lblContraseña.Text = Server.UrlDecode(Request.QueryString["clave"]);
}
```

Tanto en la inserción como en la recuperación de los datos agregados a la URI es necesario realizar un proceso de conversión previo. Su finalidad es convertir los datos de modo que no contengan caracteres no permitidos en una URL. El proceso de codificación y decodificación se lleva acabo mediante los métodos *URLEncode()* y *URLDecode()* miembros del objeto *Server* de la página.

Limitaciones de las QueryString:

La transferencia de datos entre páginas mediante el uso de *QueryString* presenta dos graves problemas que deben tenerse presentes a la hora de utilizarlas:

- 1.- Sólo pueden almacenarse pequeños datos debido a la limitación de longitud de las URLs .
- 2.- Los datos son visibles y pueden ser aprovechados por usuarios maliciosos.

5.1.4.- Recuperación información de cliente

Las peticiones HTTP también incluyen información de utilidad sobre el usuario origen de la petición como su dirección IP en internet, el nombre y versión del navegador que utiliza, detalles de la configuración de pantalla... etc. Estos datos suelen emplearse para personalizar el aspecto de la página en función del navegador o tamaño de pantalla del usuario.

Estas informaciones se encuentran organizadas en pares clave-valor contenidos en una colección de tipo `NameValueCollection` accesible a través de la propiedad `ServerVariables` del objeto `Request`. Sin embargo, también es posible acceder algunas de las informaciones de forma directa a través de ciertas propiedades. Ejemplo:

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Página sin título</title>
</head>
<body>
  <form id="form1" runat="server">
    <h3>INFORMACION DEL USUARIO</h3>
    DNS del usuario: <%=Request.UserHostName %><br />
    Direccion del usuario: <%=Request.UserHostAddress%><br />
    Tipo de peticion: <%=Request.HttpMethod %><br />
    <h3>INFORMACION DEL NAVEGADOR DEL CLIENTE</h3>
    Navegador: <%=Request.Browser.Browser%><br />
    Version: <%=Request.Browser.Version %><br />
    <h3>PERMISOS DEL NAVEGADOR</h3>
    Admite Cookies: <%= (Request.Browser.Cookies)? "SI": "NO" %><br />
    Admite Javascript: <%= (Request.Browser.Javascript)? "SI": "NO" %><br />
    Admite Frames <%= (Request.Browser.Frames)? "SI": "NO" %><br />
    Admite Hojas de estilo CSS: <%= (Request.Browser.SupportsCss)? "SI": "NO" %><br />
    Dispositivo Movil: <%= (Request.Browser.IsMobileDevice)? "SI": "NO" %><br />
    <h3>CONFIGURACION DE PANTALLA DEL USUARIO</h3>
    Resolucion: <%=Request.Browser.ScreenPixelWidth%> x <%=Request.Browser.ScreenPixelHeight%>
    pixels aprox.<br />
    Colores: <%=Request.Browser.ScreenBitDepth%> bpp
  </form>
</body>
</html>
```


5.2.- EL OBJETO RESPONSE (*HttpResponse*)

El objeto Response representa la página HTML que se genera en el servidor al procesarse una página ASP.NET, y después se envía al navegador del usuario mediante una respuesta HTTP. Este es un proceso acumulativo en el que cada control Web inscribe mediante el objeto Response su propio código HTML al procesarse, dando lugar a la página que recibe el usuario.

El objeto Response puede emplearse para añadir manualmente código a la respuesta del usuario. La clase HttpResponse posee el método Write que permite añadir una cadena de texto al código HTML de respuesta para el usuario:

Página Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Capital: </title>
</head>
<body>
<form id="form1" runat="server">
<% Response.Write("HOLA MUNDO"); %>
</form>
</body>
</html>
```

Código HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
Capital:
</title></head>
<body>
<form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJNzgZNDMwNTMzZGS8aeBwkjl9fnknH9iw1Y198nvcNw==" />
</div>
HOLA MUNDO
</form>
</body>
</html>
```

Dependiendo cuando se ejecute el método Write, el texto aparecerá inscrito en un punto distinto de la respuesta enviada al usuario:

Código en el lado. Default.aspx.cs

```
public partial class _Default : System.Web.UI.Page {
protected void Page_Load(object sender, EventArgs e) {
Response.Write("HOLA MUNDO");
}
}
```

Código HTML

```
HOLA MUNDO
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
Capital:
</title></head>
<body>
<form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJNzgZNDMwNTMzZGS8aeBwkjl9fnknH9iw1Y198nvcNw==" />
</div>
</form>
</body>
</html>
```

En el ejemplo anterior la cadena HOLA MUNDO aparece al principio del código HTML del resto de la página. Esto es debido a que la invocación del método Write se ejecutó en el evento Load de la página, antes de que comenzase la generación de código HTML de la propia página y los controles Web.

Si deseamos inscribir un código en un punto concreto de la respuesta al usuario, lo mejor es incluir el código dentro del fichero de diseño (.aspx).

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
    String nombre = "PEDRO";
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Capital: </title>
</head>
<body>
    <form id="form1" runat="server">

        <p style="font-style:italic; font-weight:bolder">
            EL NOMBRE ES:<%= Response.Write(nombre); %>
        </p>

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

Al procesarse esta página, la invocación del método Write inscribirá el valor que tenga en ese momento la variable “nombre”, y lo hará en la misma posición donde se encuentra inscrita la instrucción. El resultado será:

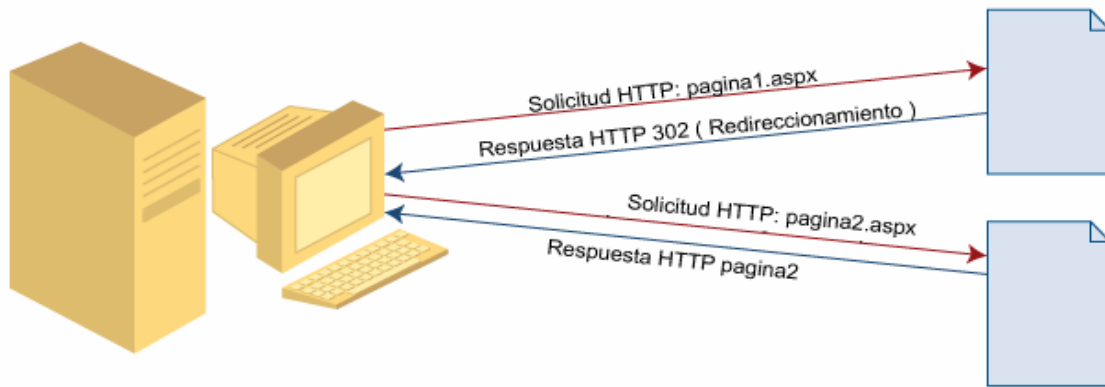
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
    Capital:
</title></head>
<body>
    <form name="form1" method="post" action="Default.aspx" id="form1">
        <div>
            <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
            value="/wEPDwUKLTczMzQ3OTE1MGRKY/Rn9YCRZFRqHZl66+LnCi9PSDY=" />
        </div>
        <p style="font-style:italic; font-weight:bolder">
            EL NOMBRE ES:PEDRO
        </p>
    </form>
</body>
</html>
```

5.2.1.- Redireccionamiento de Páginas (Response.Redirect)

Otra funcionalidad vinculada al objeto Response es el redireccionamiento de peticiones. El Redireccionamiento consiste en desviar al usuario a una dirección URL distinta de la que ha solicitado. Para ello se emplea el método *Redirect* del objeto Response.

Para realizar un redireccionamiento, la página que recibe la petición inicial debe ejecutar el método Redirect del objeto Response. Esto provoca la finalización del proceso de la página y el envío de una respuesta HTTP al navegador del usuario que le fuerza a solicitar automáticamente la URL de otra página. El proceso es totalmente transparente al usuario, que lo único que aprecia es que la dirección del navegador cambie súbitamente.



Esta operación implica la realización de dos peticiones HTTP y la pérdida de los datos de la petición HTTP original, los cuales no serán accesibles para la página destino; ya que son solicitudes distintas. Ej:

```
Response.Redirect("http://www.microsoft.com/gohere/look.htm");
```

La única manera de solventar la pérdida de los datos de la solicitud original durante el redireccionamiento, es incluirlos concatenados en la URL de destino imitando el método de envío GET de un formulario. La página de destino del redireccionamiento deberá recuperar entonces los datos empleando la propiedad QueryString del objeto Request:

Pagina1.aspx

```
public partial class _pagina1 : System.Web.UI.Page
{
    int num = 100;
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Redirect("pagina2.aspx?num=" + num.ToString());
    }
}
```

Pagina2.aspx

```
public partial class _pagina2 : System.Web.UI.Page
{
    int num;
    protected void Page_Load(object sender, EventArgs e)
    {
        num = int.Parse(Request.QueryString["num"]);
    }
}
```

5.3.- EL OBJETO SERVER (*HttpServerUtility*)

El objeto *Server* pertenece a la clase *HttpServerUtility*. Esta clase posee múltiples métodos que pueden emplearse para procesar peticiones Web de los usuarios.

Las únicas dos propiedades de esta clase son: *MachineName* que devuelven una cadena con el nombre del servidor de la aplicación Web, y *ScriptTimeout* que permite obtener y establecer el tiempo máximo de atención a un usuario inactivo. Sus dos principales métodos son *Execute* y *Transfer* que permiten la transferencia de peticiones entre páginas ASP.NET.

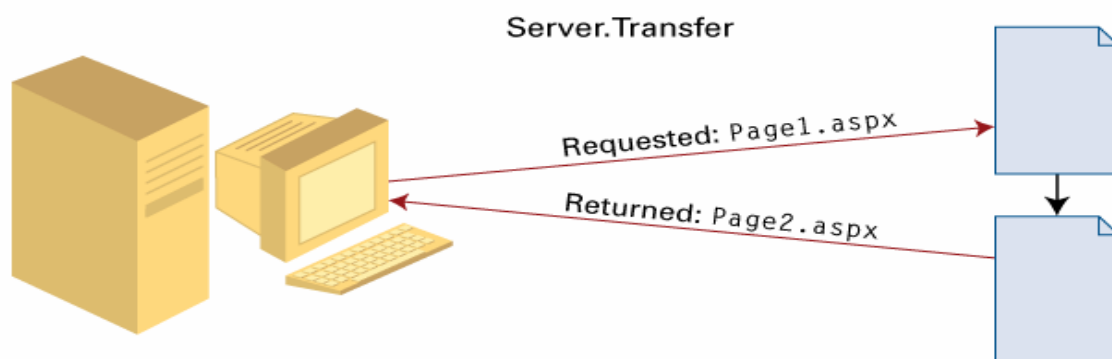
5.3.1.- Transferencia entre Páginas

En muchas ocasiones se necesita que una petición dirigida a una página ASP.NET sea respondida por otra página.

Para conseguir esto se debe transferir la petición HTTP que llega a una página hacia otra. Una vez recibida la petición por la página de destino puede ser ésta quien emita la respuesta directamente, o bien terminar retornando el control a la página inicial para que termine la respuesta al usuario. Ambas operaciones se lleva acabo invocando desde el código de servidor a los métodos *Transfer* o *Execute* del objeto *Server*.

El método *Transfer* envía la petición a la página de la URL indicada como argumento para que sea ésta última la que genere la respuesta al usuario.

```
server.Transfer("test.aspx");
```



La transferencia de petición es una operación que se hace en el servidor sin intervención del navegador del usuario. Por lo tanto; éste ni detecta el cambio y continúa mostrando la URL de la página solicitada. Esto puede alterar el funcionamiento esperado de los botones de navegación “Adelante” y “Atrás” en los navegadores convencionales.

El método *Execute* permite desviar el proceso de la página actual a la página de la URL indicada como argumento. Sin embargo; a diferencia del método *Transfer*; una vez terminado el proceso de la página secundaria, la ejecución se reanuda en la página inicial a partir de la siguiente línea de código:

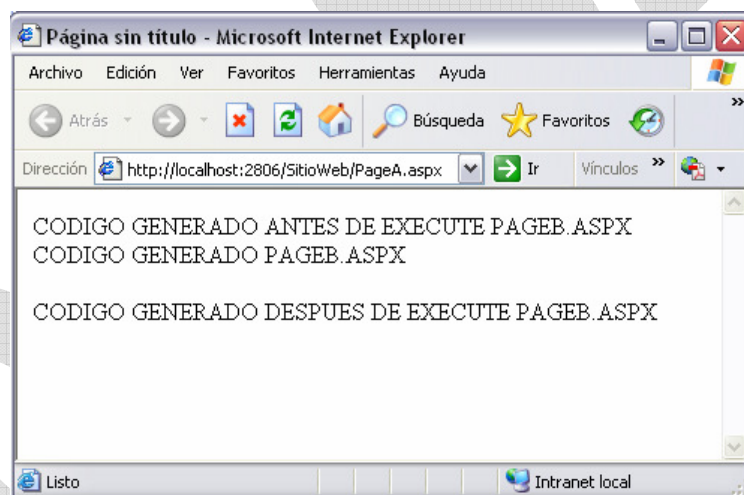
Fichero de código PageA.aspx.cs

```
public partial class PageA : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("CODIGO GENERADO ANTES DE EXECUTE PAGEB.ASPX <br />");
        Server.Execute("PageB.aspx");
        Response.Write("CODIGO GENERADO DESPUES DE EXECUTE PAGEB.ASPX <br />");
    }
}
```

Fichero de código PageB.aspx.cs

```
public partial class PageB : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("CODIGO GENERADO PAGEB.ASPX <br />");
    }
}
```

Al ejecutar una petición de la página PageA, la respuesta recibida refleja como la pagina PageB.aspx ha sido ejecutada y su respuesta embebida dentro del código de respuesta de la propia página PageA.aspx.



En ambos casos debe destacarse que los datos recibidos en la petición de la primera página, estarán también disponibles para la segunda.

También es posible acceder a los valores de los controles contenidos en página inicial desde la página secundaria empleando la propiedad *Handler* del objeto *Context*.

<pre>Public partial class Pagina1 : Page { public int Number; this.Server.Transfer("Pagina2.aspx"); }</pre>	<pre>Public partial Class Pagina2: Page { Pagina1 pg = (Pagina1)Context.Handler; pg.Number; }</pre>
---	--

CLIPSA

El fichero web.config

Todas las aplicaciones ASP.NET incluyen un fichero de texto llamado *web.config* en el que se incluyen parámetros de configuración para el servidor y la propia aplicación. Este archivo se encuentra dentro de la carpeta raíz de la aplicación en el servidor Web.

Este archivo nunca está bloqueado, por lo que puede ser modificado en cualquier momento incluso mientras la aplicación Web está funcionando. Las peticiones en curso se atenderán según los parámetros antiguos, y las nuevas según los nuevos.

La información de configuración emplea un formato predefinido en XML. El uso de XML implica la organización jerárquica de la información en etiquetas de apertura y cierre (tags). El formato XML es *case-sensitive* (distingue entre mayúsculas y minúsculas), y en el caso concreto del archivo web.config; todas las etiquetas están declaradas dentro de la etiqueta raíz **<configuration>**.

Ejemplo:

```
<?xml version="1.0" ?>
<configuration>
  <configSections>...</configSections>
  <appSettings>...</appSettings>
  <connectionStrings>...</connectionStrings>
  <system.web>...</system.web>
  <system.codedom>...</system.codedom>
  <system.webServer>...</system.webServer>
</configuration>
```

De entre todas las secciones, las más relevantes para el desarrollo de las aplicaciones Web sean las siguientes:

- **La sección <system.web>**: contiene parámetros de configuración referentes al propio servidor Web en el que se ejecuta la aplicación Web. Estos parámetros configuran múltiples aspectos del servidor tales como la autenticación, manejo de errores, modo de compilación del código... etc.
- **La sección <appSettings>**: permite definir parámetros propios de la aplicación como la cadena de conexión a una base de datos... etc. Todos los parámetros se definen según una clave que permite recuperar su valor desde el código de cualquier página ASP.NET de la aplicación.
- **La sección <connectionStrings>**: permite definir cadenas de conexión a bases de datos para su uso en la aplicación Web

ASP.NET emplea un sistema de configuración multinivel que permite especificar diferentes configuraciones para una aplicación Web.

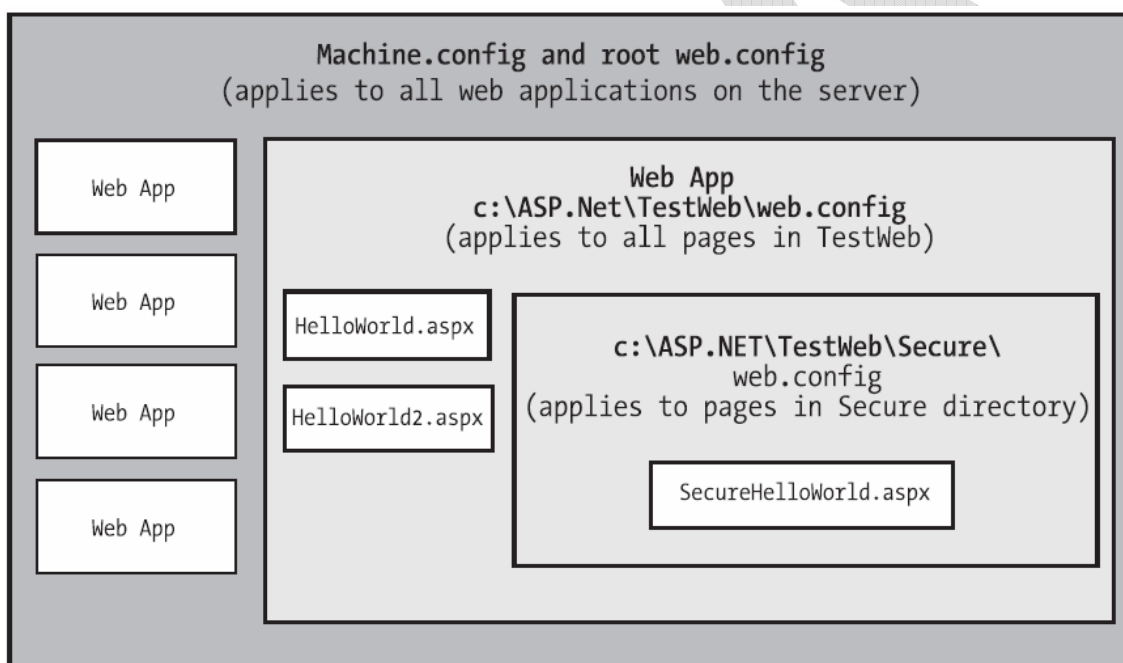
Todos los servidores Web poseen unas configuraciones predeterminadas fijadas en dos archivos denominados *machine.config* y *web.config* presentes en:

c:\Windows\Microsoft.NET\Framework\v2.0.50727\Config directory..

Los archivos *machine.config* y *web.config* situadas en ese directorio no pueden ser modificados manualmente ya que afectan a todas las aplicaciones .NET que se ejecutan en el ordenador. Sin embargo; la modificación manual del archivo *web.config* dentro de la carpeta de la aplicación Web, puede añadir o sobrescribir la configuración predeterminada de los otros dos archivos.

Dentro de una misma aplicación Web se pueden establecer diferentes configuraciones. Para ello se deben distribuir la aplicación en diferentes subdirectorios, en cada una de las cuales se puede indicar un archivo *web.config* diferente. Los parámetros de este archivo se aplicarán a todos los elementos de la aplicación Web dentro de la carpeta.

En ausencia de un archivo *web.config* en un subdirectorio, éste hereda la configuración del directorio padre correspondiente. Si por el contrario existe un archivo *web.config*; todos los parámetros no indicados se heredan de la configuración del directorio inmediatamente superior.



En el diagrama se muestran tres capas típicas de configuraciones:

- **Nivel de servidor.** La configuración se establece en los ficheros *machine.config* y *web.config* del equipo servidor y afecta a todas las aplicaciones Web en el equipo.
- **Nivel de aplicación Web:** La configuración reside en el archivo *web.config* situado en el directorio virtual de la aplicación Web.
- **Nivel de subcarpeta en la aplicación Web:** La configuración reside en el archivo *web.config* situado en la propia subcarpeta, heredando todos los parámetros no modificados de la configuración del directorio virtual de la aplicación Web.

Una de las principales utilidades del uso de este mecanismo es poder definir diferentes configuraciones de seguridad (más o menos restrictivas) en diferentes carpetas donde se sitúan archivos o páginas más delicadas.

6.1.- PARAMETROS DE SECCION <appSettings>

Esta sección se utiliza para definir valores constantes que se pueden recuperar por código desde cualquier página de la aplicación Web.

Los valores se introducen mediante la etiqueta <add> seguida de los atributos *key* que contiene la clave que identifica, y *value* que contiene el valor propiamente:

```
<appSettings>
  <add key="sqlConn" value="Server=myPc;Database=Northwind" />
  <add key="smtpServer" value="smtp.mydomain.com" />
</appSettings>
```

En este caso se han definido dos datos identificados con las claves “sqlConn” y “smtpServer”. Las claves son necesarias para poder recuperar los valores desde código.

La clase *WebConfigurationManager* se encuentra definida dentro del espacio de nombres (*System.Web.Configurat*io), posee la propiedad estática *AppSettings* que hace referencia a una colección *NameValueCollection* con todos los datos declarados en la sección. Para obtener el valor de cada dato se necesita su clave identificativa:

```
private String cadenaConexion = WebConfigurationManager.AppSettings["sqlConn"];
```

Para más información consultar MSDN: [http://msdn.microsoft.com/es-es/library/ms228154\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/ms228154(VS.80).aspx)

6.2.- PARAMETROS DE SECCION <connectionStrings>

Esta es una sección especializada en almacenar cadenas de conexión a bases de datos y delimitada por etiquetas <connectionStrings>..</connectionStrings>.

```
<connectionStrings>
  <add name="Northwind"
        connectionString="Data Source=(local);Initial Catalog=Northwind;uid=sa;pwd="
        providerName="SqlClient"/>
</connectionStrings>
```

Las cadenas de conexión se añaden a la sección mediante etiquetas <add>, de igual manera que en la sección AppSettings. Los atributos principales son los siguientes:

- **name:** Contiene una clave que identifica a la cadena de conexión para recuperarla desde código.
- **connectionString:** Contiene la cadena de texto con la cadena de conexión.
- **providerName:** Contiene el nombre del driver proveedor de datos empleado por la conexión.

La clase *ConfigurationManager* posee la propiedad estática *ConnectionString* que hace referencia a una colección *ConnectionStringSettingsCollection*. Esta colección almacena instancias de la clase *ConnectionStringSettings* entre cuyas propiedades se encuentran *name*, *connectionString*, *providerName*. Ej:

```
String cadenaConexion =
    ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
```

Para más información consultar MSDN: [http://msdn.microsoft.com/es-es/library/bf7sd233\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/bf7sd233(VS.80).aspx)

6.3.- PARAMETROS DE SECCION <system.web>

El listado completo de todos los elementos de configuración del bloque <system.web> y sus valores pueden consultarse en la página del MSDN: [http://msdn.microsoft.com/es-es/library/dayb112d\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/dayb112d(VS.80).aspx)

→ **<authentication>**: Indica el tipo de autenticación empleada por la aplicación Web para su ejecución en el servidor. La etiqueta únicamente posee el atributo *mode* que admite los siguientes posibles valores:

- **“Windows”**: Especifica que la autenticación de acceso a la aplicación Web se delega en la cuenta de acceso configurada en el IIS. Por defecto, esta cuenta es IUSR_XXX (cuenta de acceso de usuario anónimo para IIS).
- **“Forms”**: Especifica la autenticación de acceso a la aplicación Web basada en un formulario personalizado. En este caso es la aplicación quien acepta o rechaza el acceso del usuario empleando una página con un formulario de autenticación.
- **“Passport”**. Especifica la autenticación basada en cuentas de Microsoft Passport.
- **“None”**. No hay autenticación, ni usuarios ni permisos. Sólo se esperan usuarios anónimos.

Ejemplo:

```
<authentication mode="windows" />
```

El empleo de esta etiqueta se trata en mayor profundidad en la sección dedicada a autenticación.

→ **<authorization>**: Permite rechazar o aceptar el acceso a la aplicación Web a ciertos usuarios o grupos de usuarios. Para ello se emplean dos subetiquetas:

- **<deny>** : Indica un usuario o grupo de usuarios sin permiso de acceso a la aplicación.
- **<allow>**: Indica un usuario o grupo de usuarios con permiso de acceso a la aplicación.

Ambas etiquetas poseen el *roles* que permite indicar el nombre de un grupo, y el atributo *users* que permite indicar el nombre de un usuario. Los valores admitidos por el atributo *users* incluye los caracteres: ‘?’ que identifica a todos los usuarios anónimos, y ‘*’ que incluye a cualquier usuario de cualquier grupo.

```
<authorization>
  <allow roles="Administrators,Users" />
  <deny users="*" />           // resto rechazados.
</authorization>
```

Es importante tener en cuenta que la comprobación de permisos se hace siguiendo al orden de etiquetas *allow* y *deny*. De esta manera; en el ejemplo se permite el acceso a las cuentas del grupo administrador y usuarios, pero rechaza cualquier otra cuenta.

→ **<compilation>**: Permite configurar parámetros relativos a la compilación de la aplicación Web. . Esta etiqueta admite múltiples atributos pero los más comunes son: *defaultLanguage* que permite indicar el lenguaje empleado en el código de servidor (Visual Basic, o C#), y *debug* para activar la visualización de información para depuración de errores.

```
<compilation defaultLanguage="c#" debug="true"/>
```

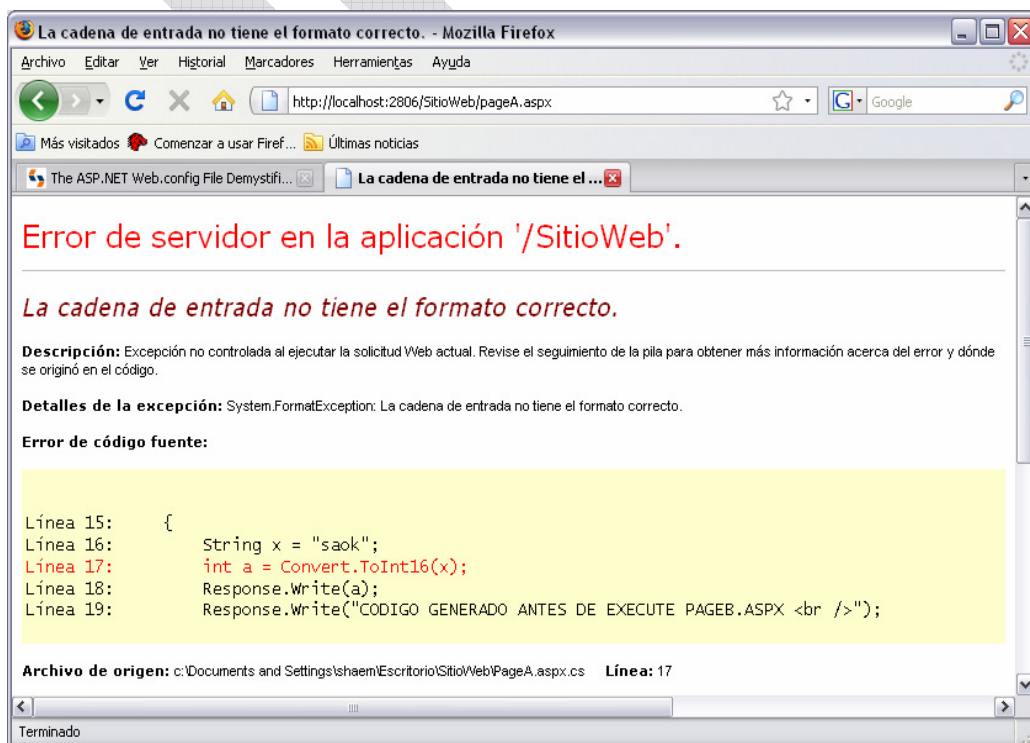
→ **<customErrors>**: Permite indicar el uso de páginas de error personalizadas que sustituyan a las páginas de error de ASP.NET. La etiqueta posee los atributos *defaultRedirect*, y *mode*. El atributo *mode* admite tres valores posibles:

- **On**: Las páginas de error personalizadas se muestran a todos los usuarios de la aplicación Web.
- **RemoteOnly**: Las páginas de error personalizadas se muestran sólo a los usuarios remotos. Los usuarios locales verán las páginas de error por defecto.
- **Off**: Todos los usuarios verán las páginas de error por defecto.

El atributo *defaultRedirect* permite indicar la URL (relativa o absoluta) de una página personalizada a mostrar si se produce un error en la aplicación Web.

```
<customErrors mode="RemoteOnly" defaultRedirect="/error.html">
```

Se considera aconsejable ocultar siempre las páginas de error de ASP.NET al usuario final dado que muestran información sobre el error inútil para él, siendo mejor mostrarle una simple página personalizada de error.

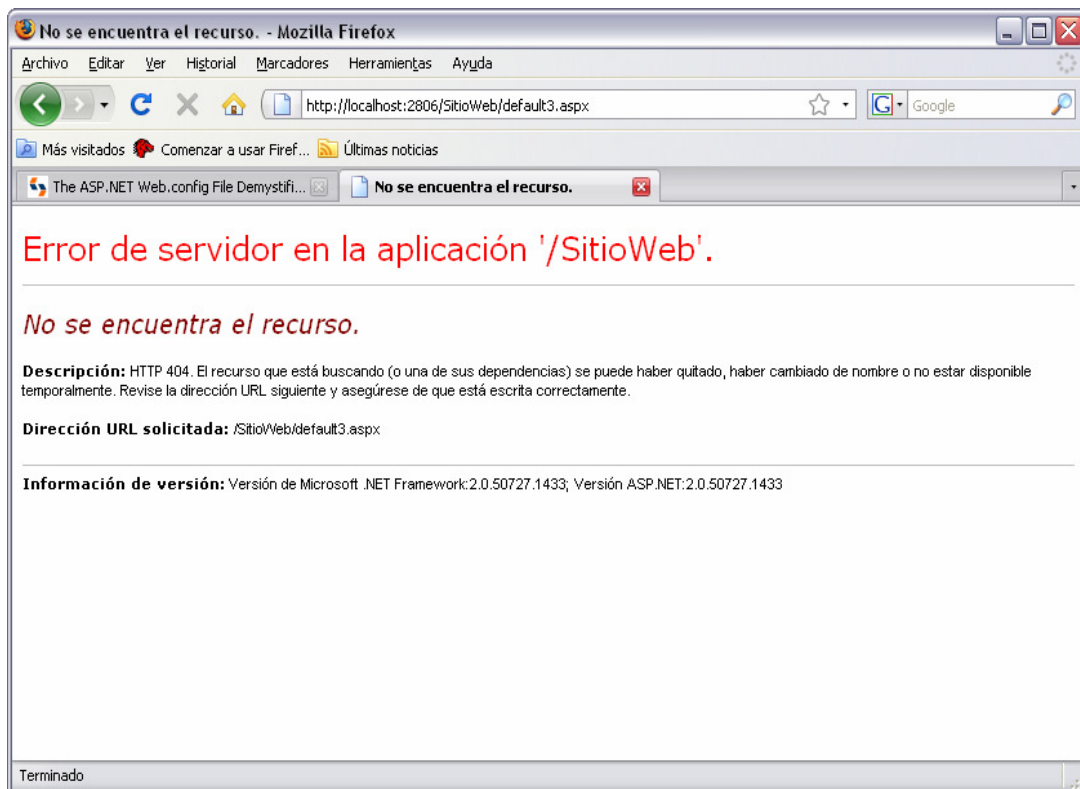


Página de error de error por defecto generada por ASP.NET en modo depuración.

Dentro de esta sección se pueden añadir etiquetas `<error>` para indicar URL de páginas personalizadas a mostrar en caso de errores HTTP. Cada etiqueta debe disponer del atributo *statusCode* con el valor de un error HTTP, y el atributo *redirect* con la URL de la página personalizada para ese error.

```
<customErrors mode="RemoteOnly" defaultRedirect="/error.html">
  <error statusCode="403" redirect="/accessdenied.html" />
  <error statusCode="404" redirect="/pagenotfound.html" />
</customErrors>
```

Los errores HTTP indican errores de servidor y solicitud de usuario, tales como solicitud denegada a una página (403), o solicitud de una página no existente en el servidor. (404)



Página de error HTTP por defecto. (Error HTTP404: Página no encontrada)

→ ***<globalization>***: Esta etiqueta permite cambiar la referencia cultural de la aplicación Web y el formato de codificación de peticiones y respuestas. El formato de codificación del texto de peticiones y respuestas se indican mediante los atributos *requestEncoding*, y *responseEncoding* respectivamente. En ambos casos, el valor por defecto UTF-8. La referencia cultural se indica mediante al atributo *culture*. Si no se especifica ninguna referencia cultural se emplea la configuración regional del sistema operativo.

```
<globalization requestEncoding="utf-8" responseEncoding="utf-8" culture="nl-NL" />
```

En este caso, la referencia cultural indicada es el Alemán – Netherlands. Este parámetro afecta a la representación de fechas, valores numéricos y monetarios especialmente. Los códigos de las referencias culturales para cada idioma/nacionalidad pueden consultarse en:

<http://msdn.microsoft.com/en-us/library/system.globalization.cultureinfo.aspx>

→ **<httpRuntime>**: Permite configurar ciertos parámetros relativos a la ejecución de la aplicación Web dentro del servidor como el número máximo de peticiones simultáneas que la aplicación atiende y el tiempo máximo de atención a un usuario inactivo. Estos parámetros pueden configurarse mediante los atributos *appRequestQueueLimit* y *executionTimeout* respectivamente.

```
<httpRuntime appRequestQueueLimit="100" executionTimeout="600" />
```

En este caso, la aplicación Web no atenderá más de cien usuarios al mismo tiempo. Si se excediese este límite el servidor emite el error “*HTTP503: Servidor Lleno*” para el resto de peticiones. De igual manera, el servidor deja de atender a un usuario si este permanece más de 600 segundos inactivo

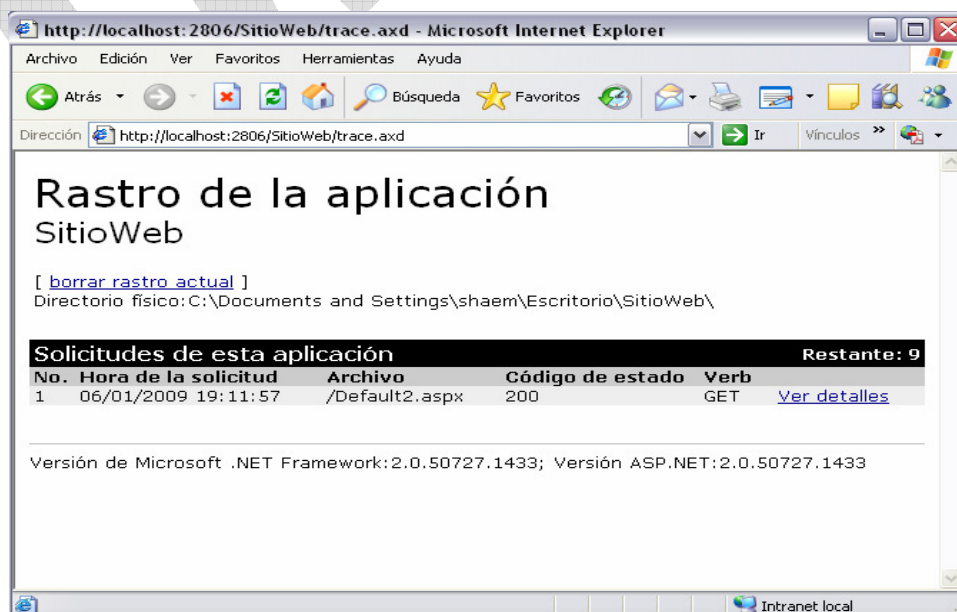
→ **<trace>**: Permite configurar la generación de trazas para la aplicación Web. Las trazas muestran información de la petición, procesado, y respuesta de cada página solicitada. Se emplea con fines de depuración y detección de errores.

Estas informaciones se registran en un archivo de registro *trace.axd* situado en el directorio raíz de la aplicación Web. No obstante, también es posible hacer que la información se muestre al final de cada página solicitada. La etiqueta posee tres atributos:

- **enabled**: Habilita o deshabilita la generación de trazas.
- **localOnly**: Permite habilitar la visualización de trazas únicamente en local.
- **pageOutput**: Permite añadir la información de la traza de cada página al final de la misma.

```
<trace enabled="true" localOnly="true" pageOutput="false" />
```

En este caso, se ha habilitado la generación de trazas de modo que no se muestren en las páginas solicitadas, y sólo sean accesibles en local accediendo directamente a *trace.axd*:

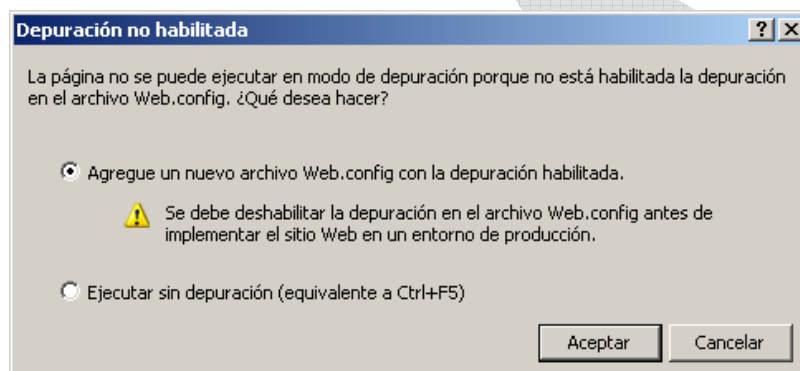


6.4.- ADMINISTRACION DEL ARCHIVO WEB.CONFIG

Las aplicaciones Web creadas Visual Studio suelen incluir un fichero de configuración web.config. En caso de no haber ningún archivo de configuración presente; la aplicación Web funciona basándose en la configuración general de .NET en el equipo servidor.

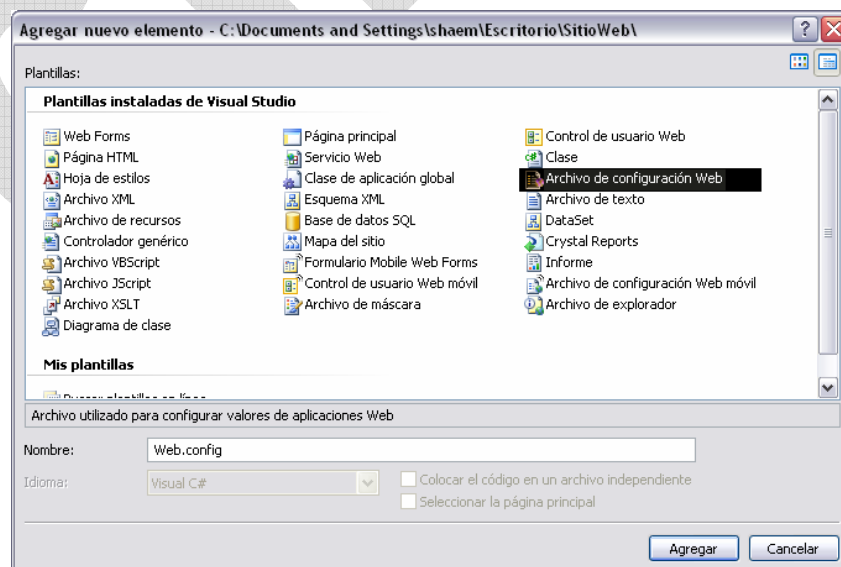
La configuración general de .NET se establece dentro del fichero *machine.config*. Este archivo define los parámetros de funcionamiento de todas las aplicaciones .NET que se ejecuten en el equipo en cuestión, y se sitúa dentro del directorio de instalación del Framework .NET.

El fichero web.config puede ser generado por el Visual Studio al iniciar una aplicación Web en modo depuración dentro del entorno de desarrollo:



Y también podemos crearlo seleccionando la opción “Archivo de configuración Web” en el cuadro de dialogo “Agregar nuevo elemento”.

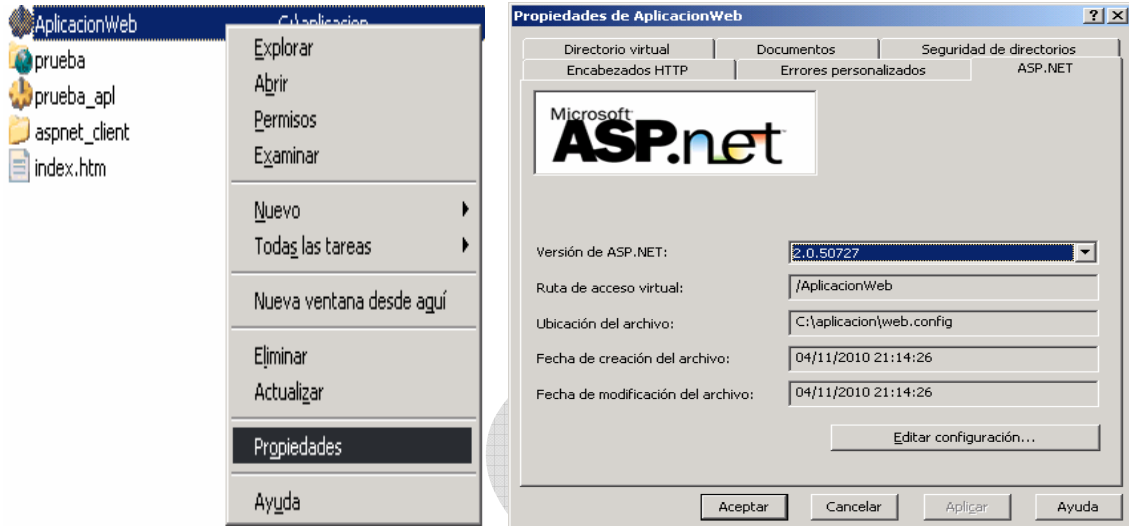
Menú Archivo → Nuevo → Archivo



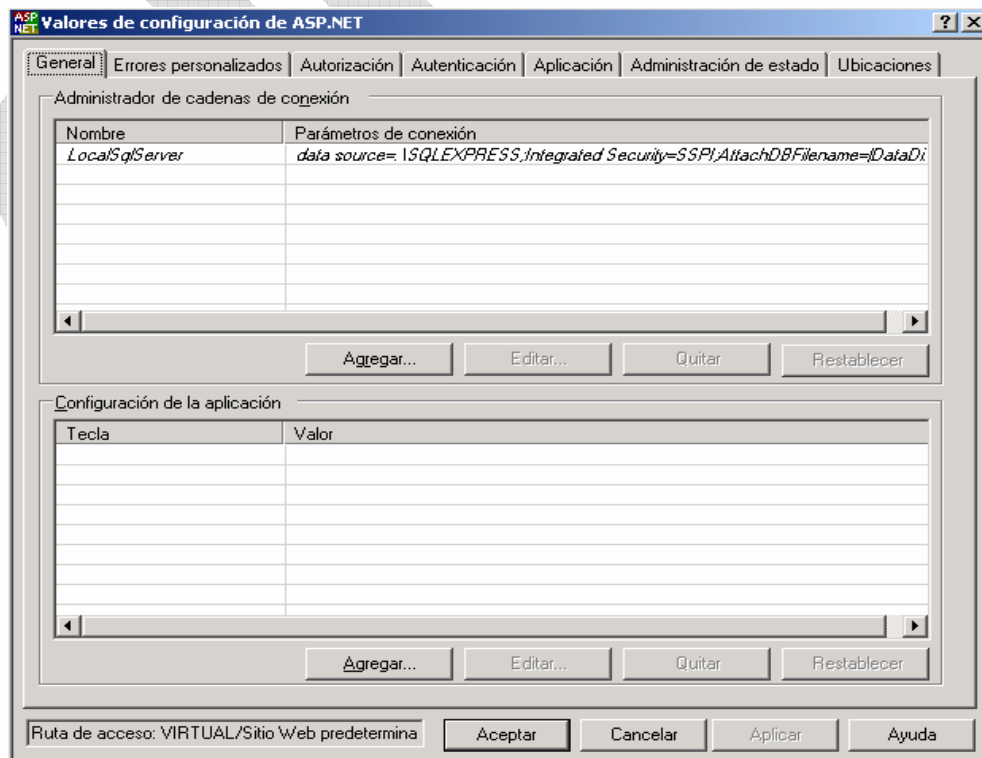
La edición del archivo web.config puede realizarse manualmente en el Visual Studio modificando el código XML.

El archivo web.config también puede generarse y/o configurarse mediante el Internet Information Services. El IIS posee un interfaz que permite editar la configuración de las aplicaciones Web que aloja.

Para acceder a la configuración de la aplicación Web, basta seleccionar propiedades en el menú contextual del directorio virtual de la aplicación, y seleccionar después la pestaña ASP.NET del cuadro de propiedades.

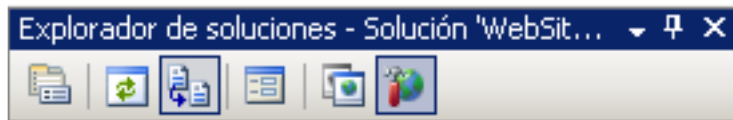


La ventana de valores de configuración ASP.NET permite editar de forma visual la diferentes secciones de la configuración de la aplicación Web. Los ajustes se codifican en el fichero web.config correspondiente.



6.5 HERRAMIENTA DE ADMINISTRACION DE SITIOS WEB

Visual Studio proporciona adicionalmente un interfaz web para la configuración del fichero de configuración de la aplicación Web. Para activarlo basta hacer clic en el botón “Configuración ASP.NET” presente en la barra de herramientas de la ventana Explorador de Soluciones.



Al pulsar este botón se lanza automáticamente una página Web con vínculos para la configuración de distintas áreas tales como: Seguridad, Configuración de aplicación, y Configuración del proveedor.

Herramienta Administración de sitios Web	
<p>Aplicación: /WebSitePerfilModelo Nombre actual del usuario: DOMINIO\ALUMNOS\ANGEL</p>	
Seguridad	Le permite configurar y editar usuarios, funciones y permisos de acceso para el sitio. El sitio está utilizando la autenticación de Windows para la administración de usuarios.
Configuración de la aplicación	Le permite administrar los valores de configuración de la aplicación.
Configuración del proveedor	Le permite especificar dónde y cómo almacenar los datos de administración utilizados por el sitio Web.

A través del vínculo “Configuración de aplicación” se puede acceder a múltiples parámetros de configuración del web.config:

Utilice esta página para configurar la aplicación con valores que no desea integrar como parte del código de las páginas, habilitar la aplicación para enviar correo electrónico, configurar la depuración, configurar una página de error predeterminada y detener o iniciar la aplicación.

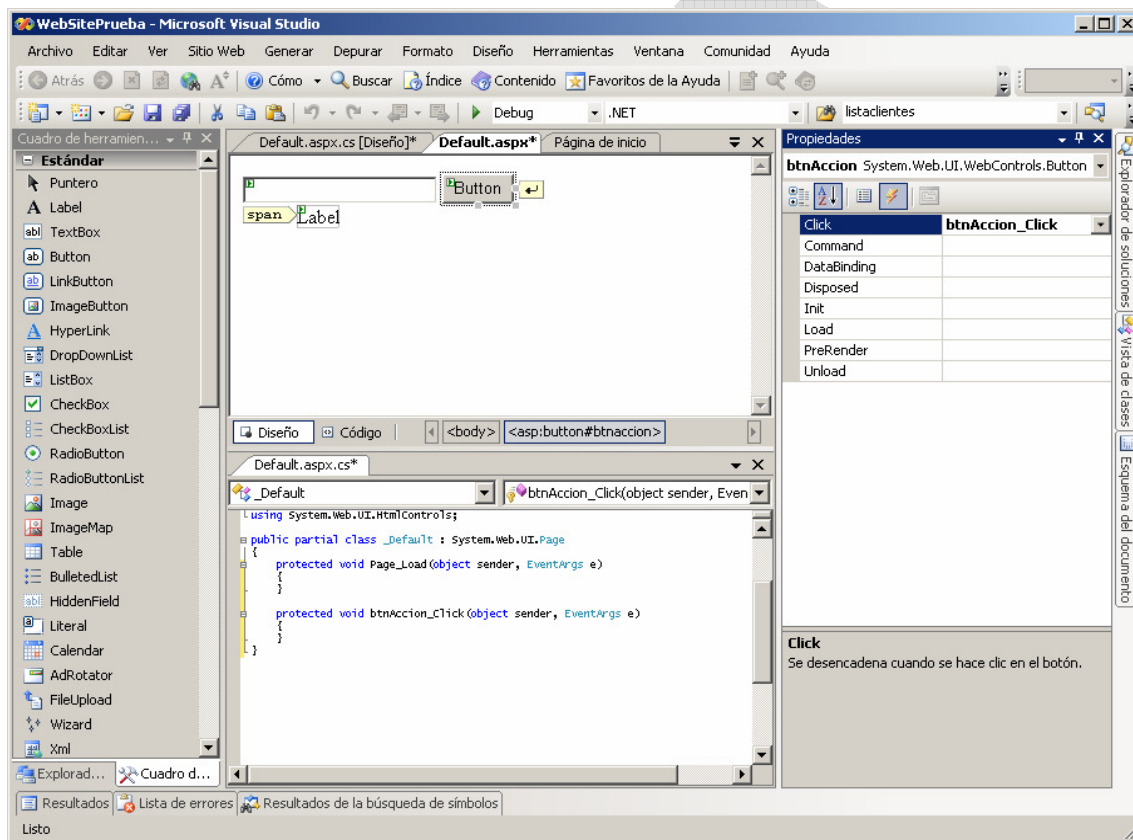
Configuración de la aplicación Configuración existente de la aplicación: 0 Crear configuración de la aplicación Administrar configuración de la aplicación	Configuración SMTP Definir configuración de correo electrónico SMTP	Estado de la aplicación La aplicación está en línea Poner la aplicación fuera de conexión
		Depuración y traza Configurar depuración y seguimiento Definir página de errores predeterminada

Práctica

Debe crearse una página ASP.NET con los siguientes elementos:

- Un control Label → lblEtiqueta
- Un control TextBox → txtNombre
- Un control Button. → btnAccion

Una vez situados los controles en la página, seleccionaremos el control btnAccion, seleccionamos el evento Click de la ventana de propiedades haciendo doble clic para que se genere el método de respuesta en el fichero de código asociado a la página. La página tendrá el siguiente aspecto:



** Notese que al añadir los controles en la página Web, no podemos situarlos donde nos dé la gana, si no que se sitúan unos en relación a otros. Esto es debido al XHTML, que no incluye información de diseño para posicionar los controles. Para poder colocar los controles siguiendo un diseño específico deberían utilizarse hojas de estilo y capas (etiquetas <div> de XHTML)*

Al hacer doble clic sobre el evento Click del botón se generará el siguiente método:

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

De igual manera; el código HTML del control Web es alterado añadiendo el evento y el nombre del método que lo atiende:

```
<asp:Button ID="btnAccion" runat="server" Text="Button" OnClick="btnAccion_Click" />
```

Supóngase que queremos que se muestre en la etiqueta lblEtiqueta el texto “Hola”, seguido del nombre escrito en la caja de texto al hacer clic en el botón btnAccion. Por ejemplo: Si escribimos por ejemplo “pedro” en la caja de texto, queremos aparezca la etiqueta mostrando el mensaje “Hola Pedro” al hacer clic sobre el botón. Además queremos también que una vez pulsado el botón btnAccion, el fondo de la caja de texto se vuelva de color gris.

Para conseguirlo basta añadir el código de lo que queremos que suceda dentro del método “btnAccion_Click”. El código de este método se ejecutará en el servidor cada vez que el usuario pulse el botón btnAccion en su navegador.

```
protected void btnAccion_Click(object sender, EventArgs e)
{
    // Asigno a la etiqueta lblEtiqueta, el texto contenido en la caja de texto txtNombre.
    lblEtiqueta.Text = "HOLA " + txtNombre.Text;
    // Cambio a la caja de texto txtNombre el color de fondo.
    txtNombre.BackColor = System.Drawing.Color.Silver;
}
```

A diferencia de lo que sucedía con los controles HTML de cliente; las propiedades de los controles Web relativas a su aspecto o diseño tales como su color de fondo, tipos de bordes, tamaños... etc; son accesibles a través de la variable de instancia que los referencia.

El código HTML de la página ASP queda así:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Página sin título</title>
</head>
<body>
<form id="form1" runat="server">
<asp:TextBox ID="txtNombre" runat="server"></asp:TextBox>
<asp:Button ID="btnAccion" runat="server" Text="Button" OnClick="btnAccion_Click" />
<br />
<asp:Label ID="lblEtiqueta" runat="server"></asp:Label>
</form>
</body>
</html>
```

Y el código del fichero de código asociado a la página:

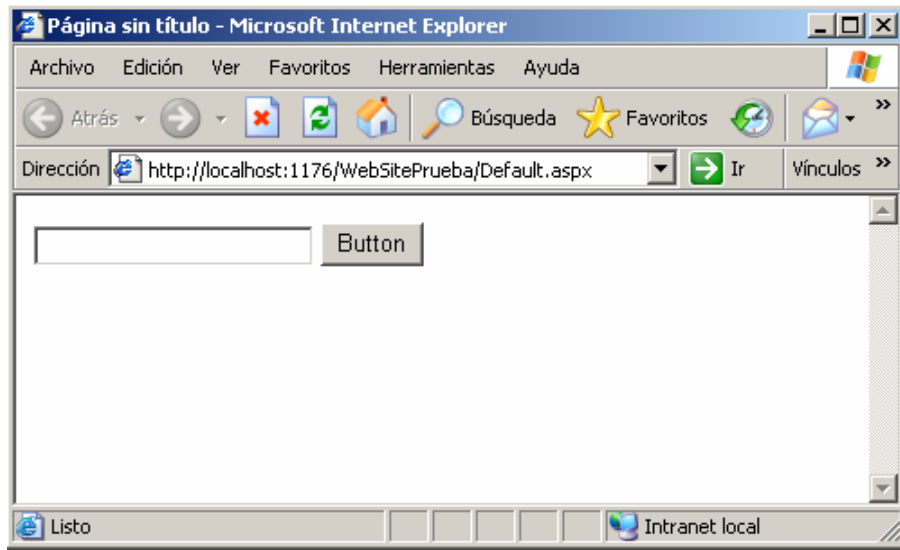
```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected void btnAccion_Click(object sender, EventArgs e)
    {
        lblEtiqueta.Text = "HOLA " + txtNombre.Text;
        txtNombre.BackColor = System.Drawing.Color.Silver;
    }
}
```

Como puede verse; los valores de las etiquetas ID presentes en el código HTML de cada control, se corresponde con el nombre de las variables de instancia que los representan en el fichero de código adjunto.

Si ejecutamos la aplicación Web, se abrirá nuestro navegador mostrando el resultado que obtiene el navegador del usuario al solicitar la página:



El código HTML que recibe el navegador desde el servidor Web es:

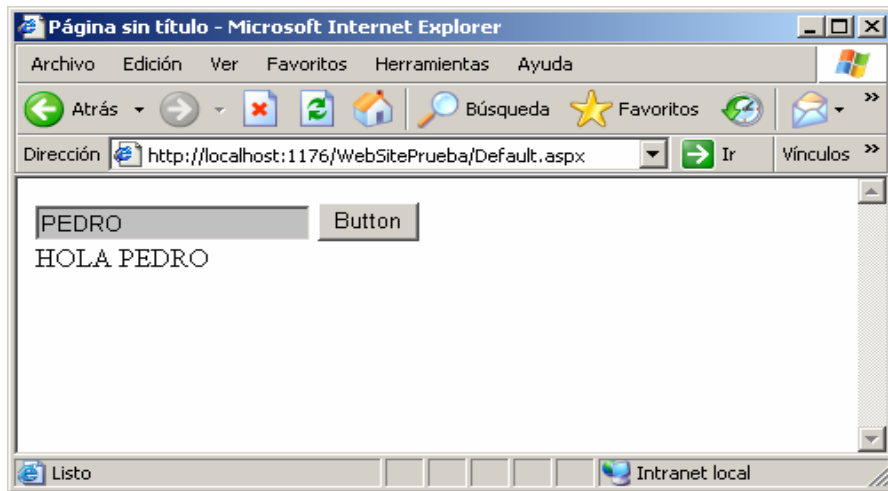
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
Página sin título
</title></head>
<body>
<form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMTU5MTA2ODYwOWRksRuirYTJ/emrRL4xYMC/qNsFRd4=" />
</div>
<input name="txtNombre" type="text" id="txtNombre" />
<input type="submit" name="btnAccion" value="Button" id="btnAccion" />
<br />
<span id="lblEtiqueta"></span>
<div>
<input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/wEWAwLy0dSRCAGIMKjDAL26aaIA8rGzasNSUU+vpu2onfPlrVO35zn" />
</div>
</form>
</body>
</html>
```

Si observamos el código veremos que no hay ni rastro de las etiquetas de los controles Web que hemos incluido: <asp:textbox>, <asp:label>... . En su lugar sólo aparecen controles HTML de cliente que mantienen sin embargo el valor del atributo Id.

La conclusión a la que debemos llegar es que los controles Web, a diferencia de los controles HTML de cliente; no se envían al navegador del cliente. Cuando se solicita y ejecuta una página ASP.NET, cada control Web genera un bloque de código combinado HTML / Javascript que se incrusta con el resto de HTML del diseño de la página y se envía al navegador Web.

Sin embargo, a pesar de esta transformación se puede distinguir qué control Web generó cada control HTML observando el valor de sus atributos ID, ya que éste se mantiene.

Si escribimos nuestro nombre en la caja de texto y pulsamos el botón `btnAccion`, el navegador vuelve a solicitar la página ASP.NET, y ésta se solicita nuevamente el servidor. Sin embargo; la página recibida tiene ahora un aspecto distinto:



El código HTML recibido por el navegador del cliente es algo distinto al recibido en la primera solicitud:

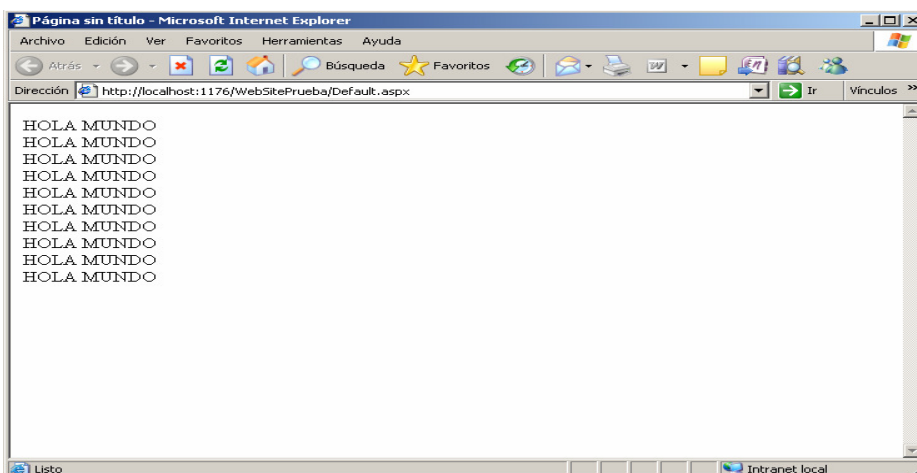
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
    Página sin título
</title></head>
<body>
    <form name="form1" method="post" action="Default.aspx" id="form1">
        <div>
            <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMTU5MTA2ODYwOQ9kFgICAw9kFgQCAQ8PFgQeCUJhY2tDb2xvcgqWAR4EXyFTQgIIZGQCBQ8PF
gIeBFRLeHQFCKhPTEEGUEVEUk9kZGSgexlMmMnw9teiziWkQ56q+7webQ==" />
            </div>
            <input name="txtNombre" type="text" value="PEDRO" id="txtNombre"
style="background-color:silver;" />
            <input type="submit" name="btnAccion" value="Button" id="btnAccion" />
            <br />
            <span id="lblEtiqueta">HOLA PEDRO</span>
        </div>
        <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/WEAWKnrCMOCQKIGMKjDAL26aaIA6Qf00ziqiPdh4s89hy1P7AZZALQ" />
        </div>
    </form>
</body>
</html>
```

Nuevamente, las secciones de HTML subrayadas representan los controles Web que están incluidos en la página ASP.NET, pero convertidos en HTML para el usuario.

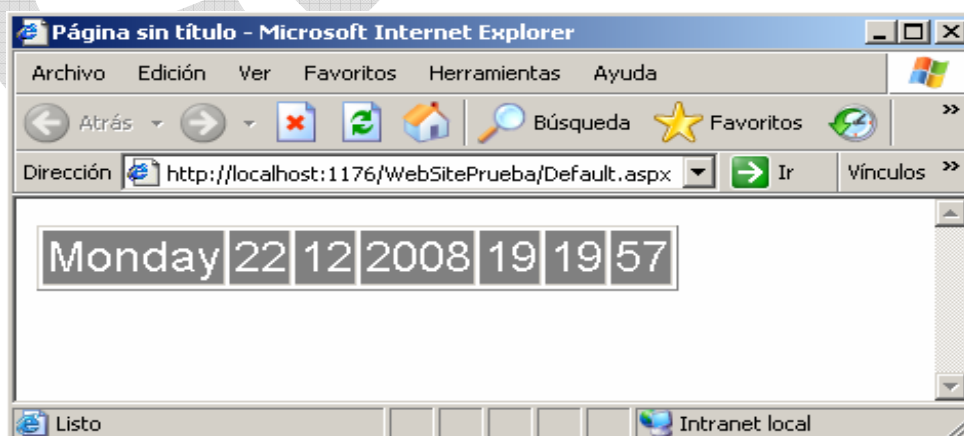
El control HTML `InputBox` que representa el control Web `TextBox` posee el valor que se introdujo en la anterior solicitud de la página pero en fondo gris (*atributo `style="background-color:silver;"`*). De igual manera; la etiqueta en línea `` que resulta del control Web `Label`; muestra como valor: "HOLA PEDRO". Todos estos cambios son el resultado de la manipulación al ejecutarse el método `btnAccion_Click` del valor de las propiedades `Text` y `BackColor` del control `txtNombre`.

EJERCICIOS

1. Crear una página ASP.NET llamada hola.aspx, que muestre el mensaje "HOLA MUNDO" repetidos diez veces. Hazlo empleando código en línea.
2. Repite el ejercicio anterior, pero esta vez emplea código en el lado. Puedes emplear el método del evento Load de la página.

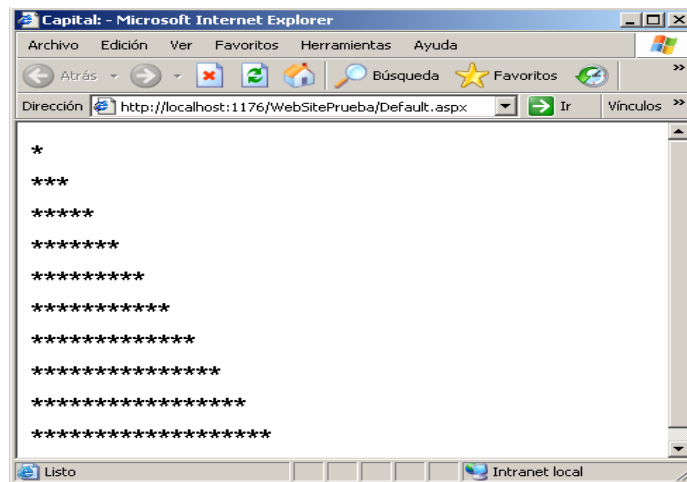


3. Crea una página ASP.NET que muestre la suma de los 100 primeros números naturales y muestre el mensaje: "La suma de los primeros 100 números naturales es: XXX". El cálculo del sumatorio debe hacerse en el código del lado de la página y la visualización mediante código en línea.
4. Crea una página ASP.NET que muestre la fecha y hora actuales. Hazlo empleando código en línea.
5. Repite el ejercicio anterior, pero esta vez emplea código en el lado.



Nota: Para modificar detalles como el color de fondo, bordes, color y tamaño de la letra en las celdas de la tabla debes emplear estilos.

6. Crea una página ASP.NET que muestre el siguiente dibujo al ser solicitada. Haz dos versiones distintas de la página; una con código en línea, y otra con código en el lado.



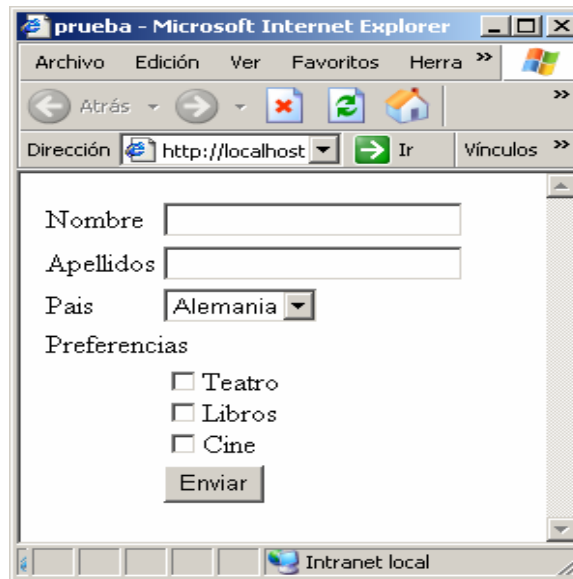
7. Crea una página ASP.NET que muestre una matriz con las multiplicaciones de todos los valores enteros comprendidos entre el 1 y el 10. El resultado debe ser semejante al siguiente:

The screenshot shows a Microsoft Internet Explorer window titled 'Capital: - Microsoft Internet Explorer'. The address bar displays 'http://localhost:1176/WebSitePrueba/Default.aspx'. The main content area displays a 10x10 multiplication table:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

The status bar at the bottom indicates 'Listo' and 'Intranet local'.

8. Crea una página HTML de nombre “form.htm” con el siguiente diseño empleando controles simples HTML:



prueba - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herra >>

Atrás > >>

Dirección <http://localhost> Ir Vínculos >>

Nombre

Apellidos

Pais

Preferencias

☐ Teatro

☐ Libros

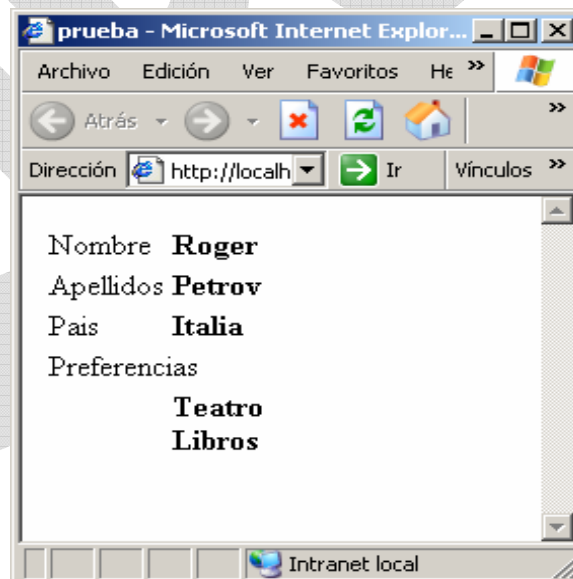
☐ Cine

Enviar

Intranet local

Al pulsar el botón “Enviar” el formulario deberá enviar los datos a la página “datos.aspx” empleando el método POST.

Diseña la página “datos.aspx” para que muestre los valores introducidos por el usuario en cada uno de los campos del formulario:



prueba - Microsoft Internet Explorer

Archivo Edición Ver Favoritos He >>

Atrás > >>

Dirección <http://localh> Ir Vínculos >>

Nombre **Roger**

Apellidos **Petrov**

Pais **Italia**

Preferencias

Teatro

Libros

Intranet local

9. Crear una página HTML con el siguiente formulario para tramitar las compras de una frutería virtual. Esta página debe llamarse "venta.htm". Para su confección debe emplearse controles HTML.

Producto

<input type="radio"/> Judias <input type="radio"/> Garbanzos <input type="radio"/> Lentejas	Cantidad <input type="text"/> Kgs	Tarjeta Visa <input type="checkbox"/>	<input type="button" value="ENVIAR DATOS"/>
---	-----------------------------------	---------------------------------------	---

El botón "ENVIAR DATOS" debe invocar la página "factura.aspx". La página debe mostrar los siguientes datos:

FACTURA

Producto: Garbanzos
 Unidades: 23
 VISA: SI

Importe: 12.50€

El cálculo del importe se hará en base a las siguientes reglas:

El Kilo de Judías vale 2€, el de garbanzos vale 2'5€, y el de lentejas vale 1'25€. Si la cantidad es mayor de 10 kilos se aplica un descuento del 2%, y si es mayor de 50 Kilos se aplica un descuento del 10%. Adicionalmente, si el pago se realiza con tarjeta VISA, se aplica un descuento adicional del 5%.

Crear dos versiones de este ejercicio empleando los métodos de envío GET y POST entre el formulario HTML y la página ASP.NET

10. Crea una página Web ASP.NET que permita introducir una cantidad de euros, y devuelva el desglose en las distintas monedas.

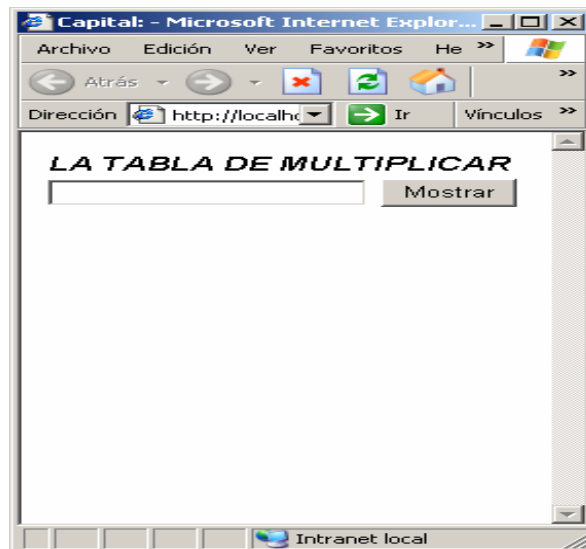
DESGLOSE DE MONEDAS

Cantidad a desglosar:

10000	<input type="text" value="0"/>
5000	<input type="text" value="2"/>
2000	<input type="text" value="0"/>
1000	<input type="text" value="0"/>
500	<input type="text" value="0"/>
200	<input type="text" value="0"/>
100	<input type="text" value="0"/>
50	<input type="text" value="0"/>
25	<input type="text" value="0"/>
10	<input type="text" value="0"/>
5	<input type="text" value="0"/>
2	<input type="text" value="0"/>
1	<input type="text" value="0"/>

Terminado

11. Crea una página Web que muestre la tabla de multiplicar de un número indicado por el usuario. Cuando el usuario solicite por primera vez la página debe mostrarse un formulario semejante el siguiente:



Al pulsar el botón “mostrar” deberá presentarse la página con la tabla de multiplicar correspondiente al valor introducido por el usuario.



12. Modificar la página Web del ejercicio anterior para que se compruebe que el valor introducido por el usuario es numérico y comprendido entre 1 y 10. En caso de no ser así; deberá mostrarse un mensaje de error indicando que el valor no es válido.

13. Repite el ejercicio anterior pero esta vez empleando varias páginas ASP.NET:

La primera página "form.aspx" contendrá el formulario en el que el usuario introduce el número del que quiere ver la tabla de multiplicar. Al pulsar el botón Mostrar; se invocará a la página "tabla.aspx" si el valor introducido es válido, en caso contrario; se mostrará la página "error.aspx".

La segunda página "tabla.aspx" mostrará los valores de la tabla de multiplicar del número seleccionado en la página anterior. Incluirá también un hipervínculo a la página "form.aspx" para volver a empezar.

La tercera página "error.aspx" mostrará un mensaje de error. Incluirá también un hipervínculo a la página "form.aspx" para poder volver a empezar.

14. Crea una página Web que muestre el siguiente formulario:

TABLA DE MUESTRAS	
muestra 1	<input type="text"/>
muestra 2	<input type="text"/>
muestra 3	<input type="text"/>
muestra 4	<input type="text"/>
muestra 5	<input type="text"/>
muestra 6	<input type="text"/>
muestra 7	<input type="text"/>
muestra 8	<input type="text"/>
muestra 9	<input type="text"/>
muestra 10	<input type="text"/>
<input type="button" value="Calcular"/>	

Al pulsar el botón "Calcular", la página debe mostrar en el lado derecho de la misma los siguientes valores:

- El máximo
- El mínimo
- El sumatorio
- La media.

15. La página debe comprobar los valores contenidos en las cajas de texto. Si alguna de las cajas de texto no contiene un valor numérico deberá aparecer con el fondo rojo. Diseñar una página web ASP.NET que muestre tres botones con los nombres "ROJO", "VERDE" y "AZUL", y cambie de color de fondo al correspondiente cuando se pulse cada uno de los botones.

Haz dos versiones de la página. Una empleando tres controles Input de HTML, y otra empleando tres controles Web Button.

16. Crea una página ASP.NET que muestre la dirección IP del usuario, y el nombre y versión del navegador que está utilizando el usuario que la solicita.