

# La Web dinámica

En este capítulo analizaremos las tecnologías disponibles para generar contenido dinámico en la Web, haremos una introducción a ASP.NET y, por último, realizaremos un repaso por el lenguaje HTML y otros conceptos que nos serán útiles en próximos capítulos.

<b>El lenguaje HTML</b>	<b>16</b>
Lograr interactividad	16
Los lenguajes dinámicos	18
.NET	23
¿Cómo funciona una página web?	25
<b>HTML</b>	<b>30</b>
Etiquetas utilizadas	33
Tablas	34
Formularios	37
<b>Resumen</b>	<b>45</b>
<b>Actividades</b>	<b>46</b>

## EL LENGUAJE HTML

HTML (*HyperText Markup Language*) es el lenguaje para escribir y generar una página de Internet. Las páginas web realizadas en HTML son estáticas, es decir, lo que se muestra en una página web HTML es exactamente igual en cualquier momento y lugar mientras no lo modifiquemos manualmente. Además, no podemos interactuar con el usuario.

Para solucionar el problema de lo estático de las páginas HTML han surgido múltiples opciones en el mercado durante la evolución de Internet. Antes de estudiar con un poco más de profundidad estas opciones, comenzaremos por ver los entes que participan en una página web.

Las páginas web son archivos informáticos que están alojados en un equipo denominado **servidor**. Éste es el encargado de enviar estos archivos a otras computadoras que lo solicitan, llamadas **clientes**. Entonces, un cliente pide, y un servidor “sirve”. ¿Quién es el cliente? Ni más ni menos que el navegador o **browser**, que, ante el pedido nuestro de visualizar una página web, realiza una petición al servidor. Éste devuelve la información y el navegador nos la muestra en el equipo.

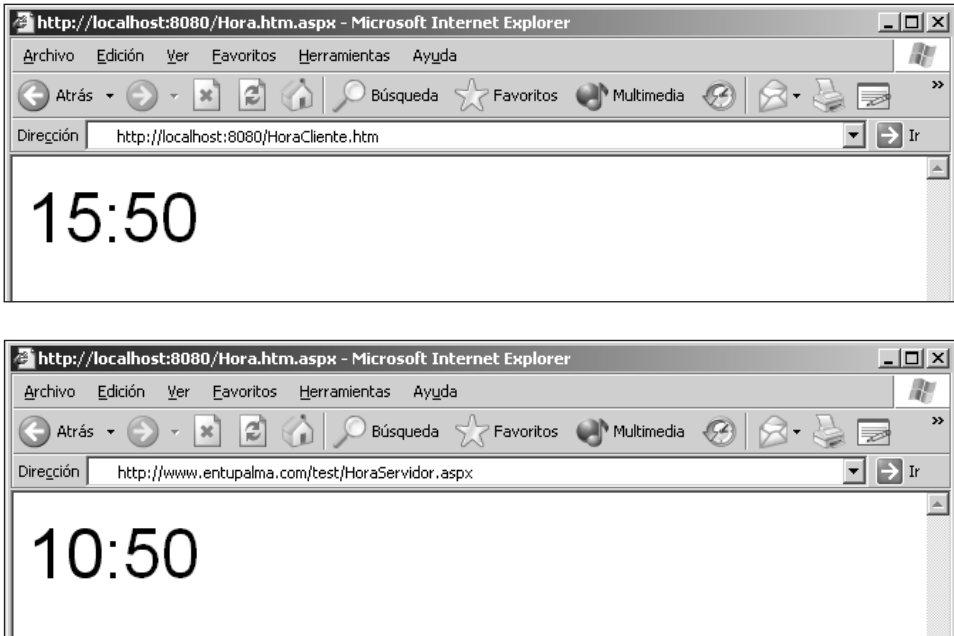
Habíamos comentado que HTML es estático, ¿qué significa esto? Imaginemos que vamos a una entidad gubernamental a realizar un trámite de impuestos. Supongamos que, ante nuestro pedido de trámite, la persona que nos atiende nos da un formulario para completar y sus instrucciones en otra hoja. Las dos páginas que nos dieron ya están preimpresas, son copias de un original que alguna vez habrán generado. Eso es HTML; es una página que ya está preimpresa, su contenido es fijo y siempre el mismo. Por supuesto que igualmente un HTML se puede cambiar, pero sería el mismo caso que cuando la entidad gubernamental cambia el formulario y lo imprime otra vez: se realiza el trabajo de modificarlo y se publica manualmente.

### Lograr interactividad

Para solucionar esta limitación que tiene el HTML han surgido tecnologías de dos tipos: las que generan contenido dinámico del lado del servidor y las que generan contenido dinámico del lado del cliente. Para el caso de las primeras, es el equipo servidor quien se encarga de ejecutar las tareas que programemos y devolver su resultado. En lo referido a las tecnologías de cliente, el programa se ejecuta en el equipo cliente, o sea, en el navegador web.

Supongamos que hacemos un programa que muestra la hora actual dentro de la página (**Figura 1**). Claramente, una página HTML estática no nos serviría, ya que deberíamos estar cambiando el archivo HTML cada minuto. Por ello debemos

decidir utilizar alguna tecnología dinámica. Y, conociendo cómo funciona Internet, digamos que nuestro servidor se encuentra alojado en otro país (algo muy frecuente en la Red). Si la aplicación que muestra la hora se ejecuta en el servidor, obtendremos la hora del país donde está alojado; en cambio, si lo hace en el cliente, obtendremos la hora local de nuestra computadora. Ésta es una primera gran diferencia entre ambas tecnologías.



**Figura 1.** Aquí podemos apreciar dos páginas que muestran exactamente lo mismo: la hora actual. Ambas páginas fueron ejecutadas en el mismo instante, sólo que una es tecnología de cliente (JavaScript), y otra, tecnología de servidor (ASPNET) en un servidor de otro país.

Ahora bien, ¿cuál es la conveniente? Para responder esta pregunta, diremos que ésta depende de la aplicación que estemos realizando. Si nuestro programa ejecuta una tarea que requiere mucho procesamiento y memoria, probablemente sea más conveniente instalarla en el servidor. No sabemos de antemano qué equipo poseerá el usuario y si será capaz de ejecutarlo con la velocidad que nosotros queremos darle. Tengamos en cuenta que siempre los servidores de Internet son computadoras de gran potencia y memoria. Ahora, si la aplicación es muy sencilla, seguramente sea preferible realizarla en el cliente, para no cargar demasiado al servidor con miles de peticiones sencillas y, además, evitar tráfico innecesario entre el cliente y el servidor. Recordemos que si hay mil usuarios visualizando una página web, el servidor deberá responder por cada uno de ellos. Si necesitamos conectarnos a bases de datos o interconectar información entre distintos usuarios, por ejemplo, sólo podremos hacerlo mediante tecnología de servidor.

Volvamos al caso de los trámites en la entidad gubernamental. ¿Qué pasaría si ahora la persona que nos da el formulario, previamente tiene que completar la fecha, un número identificador, sellarlo y firmarlo? Ya estaríamos trabajando con una tecnología dinámica. ¿Por qué? Porque ahora, cada vez que pidamos el formulario a esta persona, nos dará uno diferente, ya que tendrá distinto contenido (por lo menos, fecha y número distintos). Esto equivaldría a una tecnología dinámica de servidor; en nuestro caso, el servidor sería el empleado. Este empleado deberá ser un poco más “inteligente” que el anterior. Aquél sólo debía darnos una hoja ante una petición; ahora deberá completarla antes con algunos datos.

Lo mismo pasa con los servidores web: aquellos que soporten una tecnología dinámica serán distintos y más complejos que los que solamente se ocupen de entregar archivos HTML ante una petición.

## Los lenguajes dinámicos

Repasemos un poco los lenguajes web dinámicos que han surgido:

Del lado del servidor:

- Perl
- PHP
- ASP
- Cold Fusion
- JSP / Java Servlets
- Extensiones de FrontPage

Del lado del cliente:

- JavaScript / JScript
- Java Applets
- VBScript
- ActiveX

Para el caso de este libro nos centraremos en los lenguajes del lado del servidor.

Hace unos años surgió una interfaz llamada **CGI** (*Common Gateway Interface*), cuya misión era servir de pasarela entre la Web y otros lenguajes no preparados para ella, como C, C++, Pascal y Perl. El más flexible y utilizado fue Perl, ya que tiene funcionalidades especiales para realizar un mejor manejo de nuestras aplica-



### SIEMPRE ES HTML

Por más que utilicemos una tecnología dinámica, siempre, pero siempre, el navegador recibe una página HTML estática, ya que es el único lenguaje que sabe leer (salvando el caso de tecnologías de cliente). Esto permite compatibilidad total con cualquier navegador web aunque trabajemos con tecnologías de servidor nuevas, como, por ejemplo, ASP.NET con navegadores antiguos.

ciones en la Web. Pero CGI, con el tiempo, comenzó a perder fuerza ante otros lenguajes de script como **ASP** o **PHP**. La principal diferencia entre un script y un programa ejecutable es que el primero se interpreta en el momento de ejecutarse, es decir, siempre está el código fuente visible por el servidor y nunca existen los ejecutables compilados (los clásicos archivos **.exe** de DOS y Windows).

**ASP** es un desarrollo de Microsoft, quien utilizó la amplia distribución de su sistema operativo Windows NT y su servidor web **Internet Information Server** (IIS) para promover una versión script de su lenguaje Visual Basic que permitía programar páginas web dinámicas. Esta versión se incluyó en un Service Pack de Windows NT y se denominó ASP (*Active Server Pages*) o Páginas Activas del Servidor. Entendamos “activas” como interactivas o dinámicas.

## ASP

ASP surgió en 1996, cuando Microsoft quiso incorporarse a las tecnologías dinámicas de servidor e incluyó como parte de su servidor web **Internet Information Server** (IIS) una estructura de programación web de páginas dinámicas. El producto fue una actualización para IIS 2 y se pudo descargar como un paquete adicional desde octubre de 1996 en versión beta. Ésta fue la versión 1.0 de ASP, que, aunque no lo creamos, ya tenía gran parte de los objetos y la funcionalidad que todavía poseen las versiones actuales.

Junto con Windows NT 4 Option Pack apareció ASP 2.0, y allí comenzó a volverse un lenguaje más popular, para superar definitivamente a CGI y otros lenguajes disponibles en ese momento. Aquí incluyó soporte para SMTP (envío de e-mails) y permitió desarrollar aplicaciones de gran escala, con lo que grandes empresas comenzaron a adoptarlo para sus necesidades web.

También surgió una nueva versión de **Visual Interdev** (una herramienta de desarrollo para ASP), con gran integración con Visual Studio, por lo que se hacía cada vez más fácil programar en este lenguaje. También cabe mencionar la salida en esa época de la herramienta **Macromedia Dreamweaver Ultradev**. Era un

## III ASP.NET 2.0

Si bien ya es posible obtener algunas versiones preliminares de ASP.NET 2.0, en este libro utilizaremos las versiones 1.0 y 1.1, que son las estables actualmente. La versión 2 incorpora nuevos controles y funcionalidad, pero los conceptos son los mismos que empleamos en este libro, por lo que servirán de todas formas para una futura actualización.

excelente software de diseño de páginas web que incorporaba la funcionalidad de ASP para el manejo de bases de datos y de código dinámico. Luego de un tiempo comenzaron a salir nuevas versiones de PHP (considerado por la comunidad de programadores como la competencia directa de ASP), aparecieron Perl 5 y JSP (*Java Server Pages*), y allí ASP quedó unos pasos atrás, por lo que ya era necesario pensar en una nueva versión para no perder la batalla.

Así fue que con la salida de Windows 2000, apareció la versión 3 de ASP. Si bien no presentaba tantas mejoras en cuanto a lo que el lenguaje podía ofrecer, se optimizó mucho el rendimiento y la respuesta de las aplicaciones. ASP continuó teniendo una gran porción del mercado en cuanto a desarrollos web dinámicos.

A pesar de que ASP ya cumplía con lo que prometía, todavía se quedaba corto en algunos aspectos; sólo permitía utilizar dos lenguajes nativos, VBScript y JScript, que seguían siendo no tipados e interpretados, y además, la configuración de componentes adicionales era un trámite engorroso dentro del servidor. Así surgió **ASP.NET**, que originalmente se conoció como **ASP+**: una versión completamente nueva del lenguaje que, si bien logra bastante compatibilidad, es una reformulación de la plataforma.

Desde aquellos días hasta la actualidad, ASP recorrió mucho camino. Hoy podemos asegurar que el producto ya tiene una larga experiencia como plataforma de desarrollo web, tal como se puede apreciar en la línea de tiempo al pie de página. Con la llegada del nuevo milenio, a partir de la versión .NET, el producto dejó de llevar la cuenta numérica y comenzó nuevamente en 1.

## JavaScript y VBScript

JavaScript y VBScript se pueden utilizar como lenguajes de cliente. Esto permite generar código dinámico o procesar alguna información directamente en el navegador del cliente (que deberá soportar el lenguaje) sin tener que pasar por el servidor. Ejemplos de éstos son scripts que validan que los datos de un formulario estén completos, o un script que abre una ventana nueva con alguna información o una publicidad del sitio (**Figura 2**).

**Microsoft®****1995**

La empresa anuncia que se volcará a Internet.

**ASP****1996**

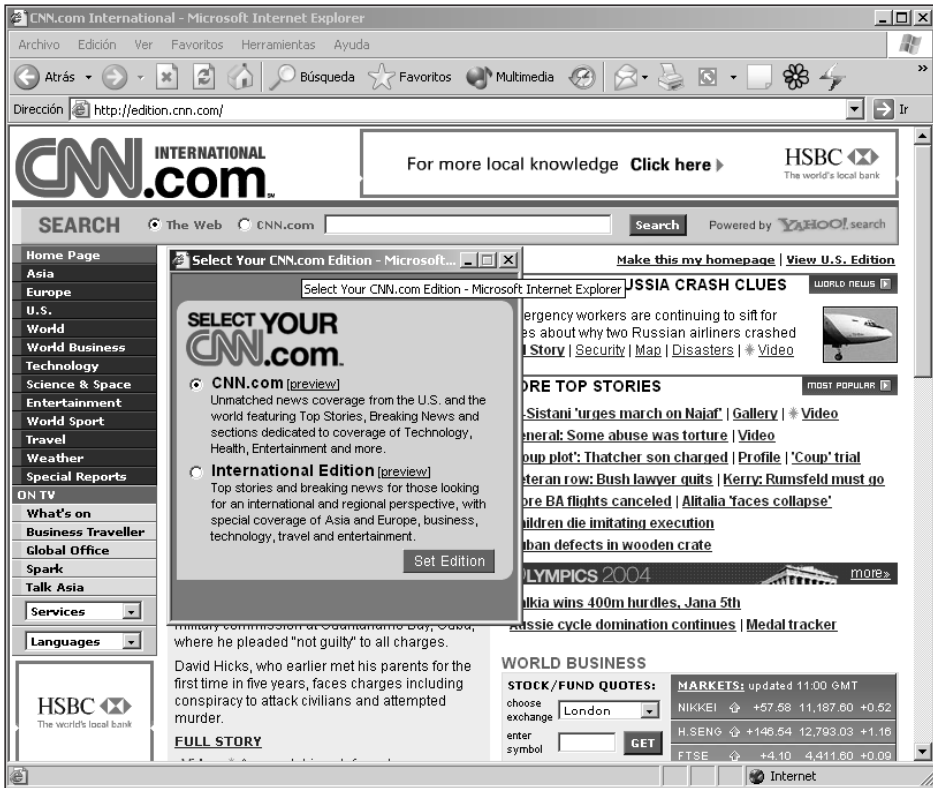
Aparece con nombre de código "Denali".

**ASP 2****1997**

La nueva versión aparece junto con IIS4.

**ASP 3****1999**

La tercera versión aparece con Windows 2000.



**Figura 2.** Con código dinámico en el cliente, es posible realizar algunas acciones imposibles de hacer con lenguaje de servidor, como el manejo avanzado de algunas funciones del navegador. En este caso vemos el típico caso de ventana “pop-up”.

Estos scripts, como dijimos, se ejecutan en el cliente y se envían incrustados en una página HTML, dentro de una etiqueta llamada SCRIPT, como la que sigue:

```
<SCRIPT LANGUAGE="Javascript">
alert('hola');
</SCRIPT>
```

**ASP.NET**

2000

Aparece la primera beta (llamada originalmente ASP+).

**ASP.NET**

2002

Es publicada la versión final.

**ASP.NET 2**

2004

Aparece la primera beta.

**ASP.NET 2**

2005

Será el año de lanzamiento de la versión definitiva.

La línea `alert('hola')` está escrita en lenguaje **JavaScript** y se ejecutará dentro del navegador del usuario. En este caso, para mostrar una ventana con el texto “hola”, como se ve en la **Figura 3**. Recordemos nada más que no todos los navegadores traen soporte de estos lenguajes. La mayoría de los navegadores para equipos PC o Mac lo incorporan, como Internet Explorer, Netscape u Opera. Sin embargo, algunos navegadores para equipos más pequeños (como equipos de mano o celulares) no sopor-

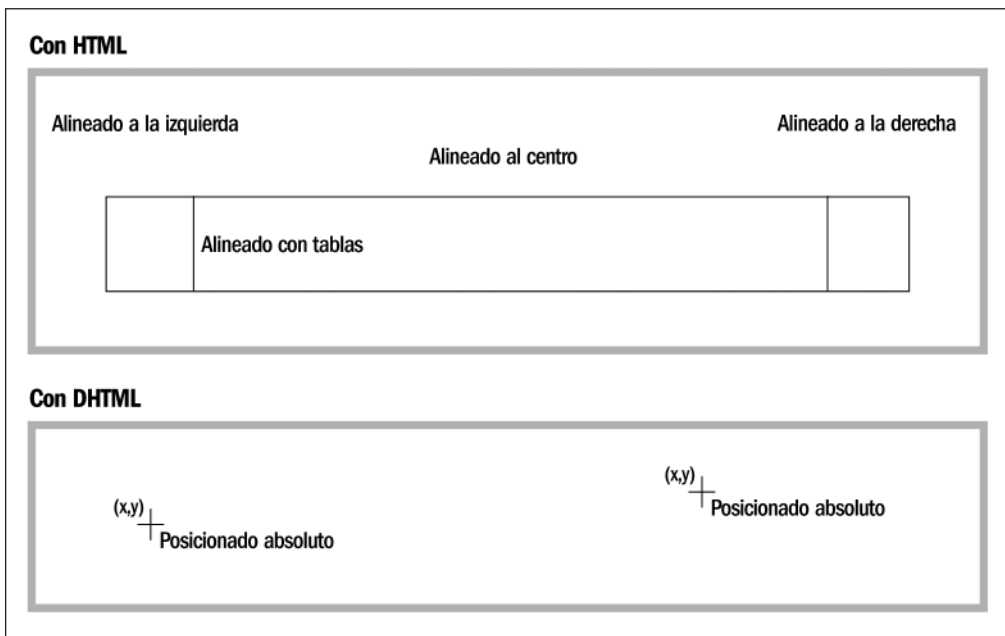


tan JavaScript, por lo que debemos tenerlo en cuenta a la hora de desarrollar nuestra página (o, si no, esperemos unos capítulos para ver cómo nos soluciona este tema ASP.NET).

**Figura 3.** El código de JavaScript `alert` permite ejecutar en el navegador cliente algo así como una ventana de información.

## Dynamic HTML

Por último, veremos algunos conceptos sobre DHTML, que no es más que una versión de HTML compatible con las versiones modernas de los navegadores que permite, mediante etiquetas HTML y código de cliente JavaScript o VBScript, generar contenido en forma dinámica o situar objetos en forma absoluta (**Figura 4**).



**Figura 4.** En este caso vemos cómo debemos ubicar los elementos en un documento HTML estándar (alineando a la izquierda, derecha o centro, o utilizando tablas) y con HTML dinámico (ubicando en posiciones absolutas).

## .NET

En los últimos años, Microsoft apuntó a un cambio radical en todas sus plataformas de desarrollo: **.NET framework**. .NET incorporó a toda la gama de desarrollo de Microsoft (incluidos Visual Basic, Visual C++ y ASP) una plataforma totalmente orientada a objetos, con decenas de clases creadas y accesibles desde cualquier lenguaje de la plataforma.

En .NET se puede programar en cualquier lenguaje compatible: VisualBasic.NET, Visual C++.NET, Visual J#.NET o el nuevo Visual C#.NET. También se pueden utilizar decenas de otros lenguajes que otras empresas incorporan al *framework* de .NET. Lo mejor de todo es que, elijamos el lenguaje que elijamos para programar, utilizaremos el mismo modelo de clases y los mismos tipos de datos compatibles. Así, ahora podemos programar una clase en un lenguaje y utilizarla en otro.

Para lograr toda esta funcionalidad, .NET trabaja con el **CLR** (*Common Language Runtime*), un entorno de ejecución operable entre aplicaciones. Este entorno es el encargado de lograr la ejecución del código, la administración de la memoria y el manejo de excepciones (errores). Es así que ahora, cualquier aplicación .NET no se compila en código ejecutable para un equipo, sino que lo hace en un lenguaje manejado intermedio denominado **MSIL** (*Microsoft Intermediate Language*). Este código luego será interpretado por el .NET framework instalado en la PC (**Figura 5**). Este proceso se asemeja al utilizado por el lenguaje Java con su **Java Virtual Machine** (JVM). Esto permite que se pueda ejecutar el código en más de un sistema operativo mientras exista un framework instalado. Hasta ahora existen frameworks estables sólo para versiones de Microsoft Windows.

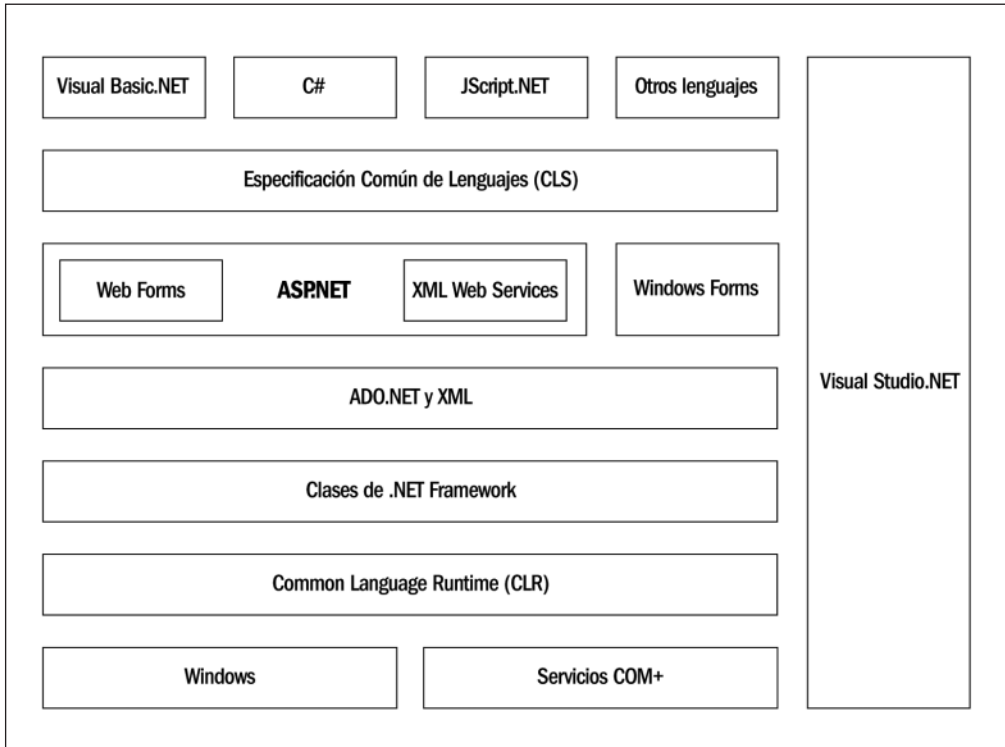
### Características

El .NET framework trabaja con el **Garbage Collector**, algo así como un “recolector de basura” que se ocupa de eliminar de la memoria aquellos objetos que ya no son utilizados por la aplicación. Esto permite que el programador no deba explícitamente liberar la memoria de un objeto al no usarse más. Este proceso es ejecutado de forma automática por .NET, lo que optimiza la aplicación.

## III CÓDIGO VISIBLE

El código fuente de lenguajes de cliente será visible por cualquier usuario desde alguna opción disponible en el navegador (como **Ver/Código Fuente** en Internet Explorer). En cambio, un lenguaje dinámico, al procesarlo en el servidor y entregar puramente HTML, no tendrá su código fuente disponible para cualquier usuario que quiera acceder a él.

Por otro lado, .NET trae soporte nativo de manejo de excepciones, lo que permite capturar distintos errores y actuar en consecuencia con cada uno de ellos. Esto da la posibilidad de procesar los errores dentro de una clase o propagarlos a quien los ha llamado, que puede ser, incluso, en lenguajes distintos.



**Figura 5.** La estructura de la arquitectura .NET para todas sus versiones. El framework incluso fue exportado en una versión compacta a equipos de mano, como handhels y smart phones.

.NET agrupa todas sus clases nativas en **namespaces** (espacios de nombre). Estos “espacios de nombre” existen en una jerarquía de clases según la función que cumplen y son accesibles desde cualquier lenguaje compatible con .NET, incluso desde aplicaciones web con ASP.NET. Esto brinda toda la funcionalidad del framework a

## POSICIONAMIENTO ABSOLUTO

Con HTML no había forma de colocar un objeto en un lugar específico de la pantalla si no era con tablas. En cambio, a través de capas o layers de DHTML, podemos ubicar un objeto en una posición fija  $(x,y)$  de la página web, y moverlo u ocultarlo mediante JavaScript o VBScript escribiendo código dinámico para cliente.

cualquier aplicación web. Esto marcó una gran diferencia ya que, hasta ahora, en ASP sólo se disponía de un pequeño conjunto de funciones.

## ASP.NET

ASP.NET no es sólo ASP 4 o una actualización al anterior ASP. Es un paradigma totalmente distinto del que se venía utilizando en el mundo web. No sólo se tendrá un acceso mayor a lenguajes con los cuales programar, como VB.NET, C#, JScript.NET y otros, sino que además permite asimilar mejor los conceptos web y ofrece una forma de programar muy similar a la que se utiliza para aplicaciones para Windows, orientada a eventos y sin los problemas que teníamos hasta ahora en el desarrollo dinámico de sitios web.

En ASP.NET se acabaron los lenguajes de script. Hasta ahora, todo el manejo se hacía en lenguajes de script difíciles de “debuguear” y con soporte de variables no tipadas (sin ningún tipo definido). Ahora se trabaja con lenguajes completos, y todos los desarrollos realizados en esta plataforma web son compilados a código manejado MSIL y ejecutados en el momento en que el servidor web necesita procesar la página web.

Ahora, con ASP.NET no hará falta instalar decenas de componentes para acceder a funciones avanzadas, gracias al soporte completo del .NET framework. Podremos enviar e-mails, procesar publicaciones de archivos, acceder a XML o generar una imagen dinámica. Mediante ADO.NET se podrá acceder de forma nativa a bases de datos de diversos proveedores en forma simple y rápida a través de “databinding” de componentes (que más adelante veremos).

## ¿Cómo funciona una página web?

Para poder ver una página web necesitamos que exista en un equipo que ejecute un servidor web. De acuerdo con la extensión del archivo que el usuario haya solicitado, éstos podrán saber si se trata de una página estática (.htm o .html) o una página dinámica (.aspx para ASP.NET o .asp para ASP clásico). Veamos algunos conceptos afines.



## .NET PARA LINUX

Mono es el nombre de un framework .NET desarrollado para Linux, actualmente en desarrollo con algunas versiones estables, pero limitadas. Sobre este framework se pueden ejecutar aplicaciones desarrolladas en .NET para aplicaciones de escritorio o para aplicaciones web en ASP.NET. Más información, en [www.go-mono.com](http://www.go-mono.com).

## Estructura

Veamos un poco cómo funciona una página web. Cuando nosotros solicitamos a nuestro navegador una dirección URL determinada, lo que estamos haciendo es pedirle al servidor que nos devuelva un archivo. Por ejemplo, cuando entramos en **www.midominio.com/index.htm** le estamos pidiendo al servidor **midominio.com** que nos entregue el archivo **index.htm** que se encuentra en el directorio raíz (/) del sitio (**Figura 6**). Cuando entramos en **www.midominio.com/empresa/quienessomos.htm**, lo que hacemos es solicitarle el archivo **quienessomos.htm**, que se encuentra en el directorio o carpeta llamada **empresa**.



**Figura 6.** Cuando solicitamos una página web, le estamos pidiendo al servidor que es parte del dominio de la URL que nos entregue el archivo indicado luego de la barra (/). Si no referimos un archivo, nos traerá el predeterminado.

Un sitio de Internet está compuesto por varios archivos, cada uno con su propia URL. Existe un archivo, en general llamado **index** o **default**, que trabaja como archivo predeterminado cuando en la URL no especificamos el que queremos. Este archivo siempre es la página de inicio del sitio. Por ejemplo, para entrar en el sitio de la **CNN**, ingresamos **www.cnn.com**. Como no especificamos un nombre de archivo luego del nombre de dominio, el servidor tomará el archivo predeterminado y nos lo entregará.

Entonces, vemos que la estructura de un sitio entero es un conjunto de archivos que se comunican a través de hipervínculos o links. Un hipervínculo es un enlace entre una página y otra que le permite al usuario recorrer todos las páginas disponibles.

## Archivos HTML

El lenguaje HTML trabaja sobre archivos de texto plano. Esto significa que el archivo contiene sólo letras, números y signos, no posee caracteres de control como otros formatos y puede ser leído por las personas sin necesidad de conversión alguna (**Figura 7**).



## OTROS LENGUAJES .NET

Cualquier empresa puede generar su propio lenguaje .NET. Es así que ya existen versiones de Delphi.NET, Cobol.NET, Python.NET, SmallTalk.NET y decenas más. Desde ASP.NET se podrá desarrollar en cualquiera de ellos. Un buen lugar donde mantenerse actualizado sobre los lenguajes disponibles es **www.gotdotnet.com/team/lang**.

```

<html><head>
<title>Yahoo!</title>
<meta http-equiv="PICS-Label" content=' (PICS-1.1 "http://www.icra.org/ratingsv02.html" l r (cz 1
<base href="http://www.yahoo.com/_ylh=X3oDMTb1c2ZmZzF2BF9TAzI3HTYxNDkEdGVzdAMwBHRtcGwDbnMtYmVOYc
</head>
<body id=bod topmargin=7 marginheight=7>
<center>
<map name=m><area alt="My Yahoo!" coords="44,0,106,47" href=r/i1><area alt=Finance coords="121,0
<tr>
<td>
<table width=100% cellpadding=5 cellspacing=0 border=0 bgcolor=#e6e6e6>
<tr>
<td align=center>
<form name=sf1 style="margin-bottom:0" action=r/zz>
<table cellpadding=0 cellspacing=0 border=0>
<tr>
<td nowrap>
<font face=arial size=-1>&nbsp;&nbsp;&nbsp; Search for: &nbsp;&nbsp;&nbsp;</font>
</td>
<td nowrap>
<font face=arial size=-1>
<input type=text size=30 name=p>&nbsp;&nbsp;&nbsp;
<select name=u>
<option value="http://search.yahoo.com/search?fr=fp-pull-web-t&p=">on the Web
<option value="http://images.search.yahoo.com/search/images?fr=fp-pull-img-t&p=">in Images
<option value="http://yp.search.yahoo.com/search/ypredirect?fr=fp-pull-yp-t&p=">in Yellow Pages
<option value="http://news.search.yahoo.com/search/news?fr=fp-pull-news-t&p=">in News
<option value="http://search.shopping.yahoo.com/search;_ylc=X3oDMTfncmo3bG5vBF9HA2dsb2JhbF9ncm91
</select>

```

**Figura 7.** Éste es el código fuente de la página HTML de [www.yahoo.com](http://www.yahoo.com). Podemos ver que, si bien hay que saber algo sobre HTML para entenderlo, es perfectamente legible.

A diferencia de los archivos en otros formatos, como ocurre con Microsoft Word, Excel, etc., donde el archivo es encapsulado en un formato propio que debe ser convertido antes de poder leerlo en pantalla, los archivos de HTML son intercambiables entre servidores de diferentes sistemas operativos, y pueden ser creados y modificados con cualquier editor de textos normal existente en cualquier sistema operativo, como lo es el Bloc de notas (**Notepad**) para Windows o el **Edit** para DOS.

Como vemos en la imagen al comienzo de la página, estos archivos tienen extensión **.htm** o **.html**, y de esta forma el servidor web identifica que se trata de una página estática y no deberá procesarla antes de entregarla al cliente.

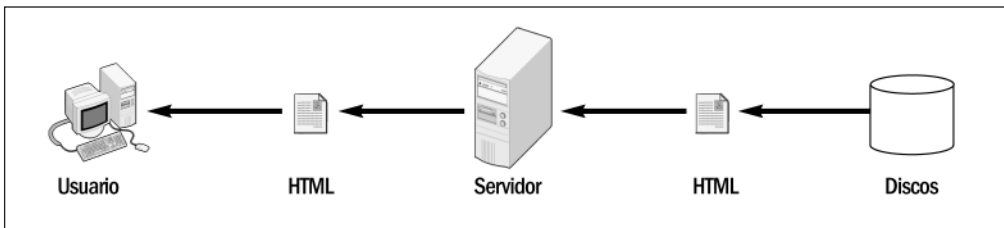
## III C SHARP

Con .NET, ahora es posible programar aplicaciones web en el nuevo lenguaje creado por Microsoft: **C#** (se lee "C sharp"). Este nuevo lenguaje es un derivado de C++ (muy similar a Java) y está especialmente preparado para el framework .NET, al permitir acceder de forma más simple a todas sus características. ASP.NET soporta este lenguaje en forma nativa.

## Sirviendo HTML

Cuando solicitamos a un servidor que nos entregue una página web estática, en formato HTML, se produce uno de los casos más simples para un servidor: éste deberá buscar si existen la carpeta y el archivo solicitado por el cliente y, en caso afirmativo, tomará el archivo y se lo entregará al cliente sin realizar ningún análisis del contenido de éste (**Figura 8**).

Este mismo proceso se genera cuando un cliente solicita algún otro tipo de archivo estático, como una imagen **GIF** o **JPEG**, una película en Flash o un documento de Word. Una ventaja de las páginas estáticas es que no es necesario que deban ser procesadas por un servidor web. Directamente pueden ser abiertas por el navegador desde el disco duro del cliente.



**Figura 8.** Un servidor no debe realizar ningún proceso cuando se le solicita un archivo HTML, más que buscarlo en sus discos y entregarlo tal cual está.

## Sirviendo ASP

Cuando solicitamos a un servidor que nos entregue una página web dinámica, desarrollada en ASP.NET, el proceso anterior se vuelve un poco más complejo. Primero, el servidor web realiza una tarea similar a la anterior. Al detectar que estamos solicitando una página en ASP.NET (cuya extensión es **.aspx**, como veremos más adelante), el servidor web realiza los siguientes pasos:

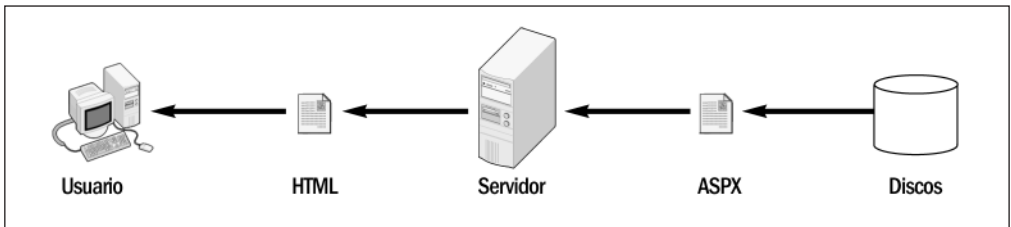
1. El Internet Information Server toma la URL que recibió del cliente y busca en su disco duro un archivo con el nombre y la carpeta solicitados.



### UNIQUE RESOURCE LOCATION

Es la dirección de una página web o recurso (imagen, archivo, etc.). Esta dirección es única en toda la Red y está compuesta básicamente por un protocolo (**http**), un dominio (**www.midominio.com**) y un archivo (**index.htm**). También podrá incluir otros objetos, como un puerto de conexión, usuario y contraseña de ingreso, o parámetros adicionales.

2. En caso de encontrarlo, realiza una serie de comprobaciones de seguridad y autenticación propias de ASP.NET (que veremos más adelante), y luego trae el archivo en cuestión.
- 3.a. Si es la primera vez que se accede a esta página ASP.NET desde que se modificó, el servidor compila el código a código MSIL usando el compilador CLR.
- 3.b. Si esta página ASP.NET ya se compiló, se pasa directamente al punto 5.
4. El código compilado se crea en una clase .NET en un archivo **.dll** ubicado en una carpeta temporal dentro del servidor, para una próxima ejecución.
5. Se instancia un objeto de la clase compilada y se le adosan todos los datos que se recibieron desde un formulario POST o datos GET para que el código procese.
6. Si el desarrollador usó Controles Web o Controles HTML (los veremos más adelante), el servidor detecta las capacidades del cliente (soporte de JavaScript, versión de navegador, marca, etc.).
7. Se ejecuta el código de la página.
8. Se envía el resultado de la ejecución (HTML) al browser cliente (**Figura 9**).



**Figura 9.** Cuando se trata de archivos dinámicos de servidor, como páginas ASP.NET, el servidor deberá procesar el documento encontrado y luego entregar el HTML resultante.

Las páginas web dinámicas no pueden ser abiertas directamente por un navegador sin ser procesadas primero por un servidor web que las ejecute. Si intentamos abrir un archivo ASP.NET en Internet Explorer, por ejemplo, veremos su código fuente y no la ejecución de éste. Por ello, para probar nuestros desarrollos deberemos utilizar un servidor web, aunque éste se encuentre instalado en nuestra misma PC.

## { } .HTM

Durante mucho tiempo se utilizó **.htm** como estándar de extensión de archivos para HTML. Esto se debe a que algunos sistemas operativos, como DOS, no soportaban más de tres caracteres en la extensión. Sin embargo, lo correcto sería utilizar **.html** para identificar este tipo de archivos.

# HTML

Ahora veamos un resumen del lenguaje **HTML** que nos permitirá entender luego el funcionamiento de ASP.NET. Como los idiomas, los lenguajes de programación tienen sus reglas y sintaxis que hay que seguir para lograr una correcta comunicación. Así como la frase “**yo ser buena persona**” está mal escrita en español, veremos muchos casos en los que el código estará mal escrito en HTML. Y mal escrito en HTML significa que los navegadores (que no son tan inteligentes como nosotros... ¿o sí?) no entenderán y no seguirán nuestras instrucciones para dibujar la página web que les ordenamos.

La unidad básica del lenguaje HTML es el **tag** (también llamado etiqueta). Un tag es una instrucción que, dentro del archivo HTML, se encierra entre paréntesis angulares (representados en las computadoras por los símbolos menor y mayor). Así, una instrucción llamada **body** se representa con el tag **<body>**. Existen tags de dos tipos:

- **De apertura solamente**
- **De apertura y cierre**

Los tags de sólo apertura se utilizan únicamente con una instrucción. Los tags de apertura y cierre permiten encerrar una cierta cantidad de datos e instrucciones entre otra instrucción mayor, y la sintaxis es la siguiente:

```
<instrucción>
.... (datos y otras instrucciones) ...
</instrucción>
```

Así, vemos que el cierre de una instrucción se realiza insertando un carácter de barra diagonal (/) antes del nombre de la instrucción.

La estructura de un archivo HTML se divide en dos partes: el encabezado (llamado **HEAD**) y el cuerpo (llamado **BODY**). Tanto el encabezado como el cuerpo deben estar insertados dentro de una instrucción de apertura y cierre denominada HTML, como en el siguiente código:

```
<html>
<head>encabezado</head>
<body>cuerpo</body>
</html>
```

Es necesario tener en cuenta que el lenguaje HTML no establece reglas en cuanto a los espacios e interlineados entre instrucciones, por lo que las siguientes expresiones son consideradas iguales:

### Ejemplo 1:

```
<tag>esto es una prueba</tag>
```

### Ejemplo 2:

```
<tag>  
esto es una prueba  
</tag>
```

El texto del tag puede estar en mayúsculas, minúsculas o mezcla de ambas, indistintamente. Queda a consideración del usuario qué opción utilizar.

## El encabezado

El encabezado de un HTML consiste en instrucciones que se insertarán dentro de un tag de apertura y cierre llamado **HEAD**, y contiene, entre otras operaciones, la instrucción **TITLE**. Esta instrucción especificará el título de nuestra página web.

Cabe aclarar que dentro del **HEAD** se puede insertar cualquier instrucción de las que veremos en los próximos capítulos, pero su función es agrupar toda la información relativa a la página web que estamos viendo y no el contenido en sí. Una gran diferencia entre el encabezado y el cuerpo es que hasta que no se descargue todo el encabezado del servidor, la página web no será mostrada. En cambio, el cuerpo es exhibido a medida que se va descargando del servidor. Esto es útil si necesitamos incluir algunos objetos que queremos estar seguros de que estarán descargados antes de que la página se dibuje.



## BARRA INVERTIDA

Hay que tener especial cuidado en no utilizar la barra diagonal invertida (\) para cerrar tags HTML, ya que el sistema no lo entenderá. La barra diagonal normal es la que termina en la esquina superior derecha. Hay que verificar su posición en cada teclado en particular.

## El cuerpo

El cuerpo de un archivo HTML es el que tendrá todo el contenido, textual, gráfico o multimedia de la página web. Todo este contenido deberá estar dentro de los tags `<body>` y `</body>`. En el próximo capítulo veremos todas las instrucciones que podremos utilizar para delinear nuestra página web.

Por ahora sólo nos quedaremos sabiendo que todo el texto contenido en el cuerpo de un archivo HTML será mostrado en la página web. Por ejemplo, vamos a ver el código de una página web muy simple que sólo nos muestra el texto **“Hola, mundo web!”** (Figura 10).

```
<html>
<head>
<title>Primer ejemplo</title>
</head>
<body>
Hola, mundo web!
</body>
</html>
```



**Figura 10.** Cuando abramos un archivo HTML en un navegador, veremos la página dibujada como el programador la diseñó (o lo más cercano a ello).

No observaremos los tags HTML, salvo que pidamos que se muestre el código fuente desde la opción **Ver/Código Fuente** de nuestro navegador.

## III INTERLÍNEA

Para lograr que el texto de una página aparezca en una nueva línea (en la página dibujada al usuario), se deberá usar el tag de nueva línea `<br>` o el de apertura y cierre de párrafo `<p></p>`. En los próximos capítulos veremos todas las instrucciones que podemos insertar en una página.

## Etiquetas utilizadas

Veamos ahora las etiquetas o tags que más usaremos en el cuerpo de una página web, para ir delineando nuestra página de acuerdo con nuestras preferencias y necesidades. Vale aclarar que así como existen tags de apertura y tags de apertura y cierre, existen algunos que pueden contener propiedades. Estas propiedades son de estilo clave/valor. O sea, a una propiedad se le asigna un valor y se expresa con la siguiente sintaxis:

```
<tag propiedad="valor" propiedad="valor">
```

## Etiquetas comunes

Éstos son ejemplos de etiquetas comunes utilizadas en HTML para distintas funciones (Figura 11).

- **Salto de línea:** `<br>`
- **Imagen:** ``
- **Negrita:** `<b>Esto es texto negrita</b>`
- **Itálica:** `<i>Esto es texto itálica</i>`



**Figura 11.** Aquí vemos en Internet Explorer cómo se diferencian los formatos y las imágenes. Con combinaciones de etiquetas HTML se crean los más complejos sitios web.

## ★ COMILLAS

Las comillas que encierran los valores de las propiedades son optativas; o sea, el navegador los reconocerá igual, aunque no las utilicemos. Pero, de todas formas, es recomendable siempre incluir las comillas para que sea considerado un documento bien formado o “well-formed”.

## Tablas

Para HTML, una tabla es una grilla formada por **filas** y **columnas**. Cada intersección entre una fila y una columna se denomina **celda**. Las celdas son los únicos elementos de una tabla que pueden contener objetos, como textos o imágenes. O sea, toda la información dentro de una tabla está definida siempre dentro de alguna celda. Aquellos familiarizados con las planillas de cálculo podrán tener una idea más clara del concepto de tabla.

La etiqueta HTML maestra que manipula las tablas es **<table>**. Con este tag de apertura y cierre podemos definir una tabla vacía:

```
<table>
</table>
```

Pero como podemos darnos cuenta, esta tabla recién creada no tiene ni filas ni columnas, o sea, no podemos crear celdas y, por lo tanto, no podemos insertar contenido. Para ello veremos otros dos tags que nos permitirán formar finalmente la grilla deseada. Ellos son **<tr>** y **<td>**, también del tipo apertura y cierre. **tr** nos permite crear una fila y **td** nos permite crear columnas por cada una de las filas.

Así, podemos formar la siguiente tabla de dos filas y dos columnas (**Figura 12**).

```
<html>
<head>
<title>Prueba de la creación de tablas</title>
</head>
<body>
<table>
<tr><td>1.1</td><td>1.2</td></tr>
<tr><td>2.1</td><td>2.2</td></tr>
```



## CÓDIGOS FUENTE

En el sitio **onweb.tectimes.com** encontrarán todos los archivos de los ejemplos que vayamos creando en todo el libro. Los hallarán separados por capítulos y con el nombre que aparece subrayado antes del código.

```

</table>
</body>
</html>

```



**Figura 12.** Aquí vemos la tabla en acción. Con algunas propiedades, como *width*, es posible definir el ancho de la tabla para que quede más amplia, o determinar también grosor del borde y color.

La tabla no tiene por qué ser cuadrada, o sea, de  $n \times n$ . Puede ser del tamaño que queramos, por ejemplo,  $1 \times 1$ ,  $2 \times 4$ ,  $10 \times 4$ ,  $150 \times 5$ , etc.

La combinación de todas estas propiedades de **table** nos permitirá lograr los efectos visuales que buscamos. A la vez, está permitido crear tablas dentro de otras tablas, llamadas tablas anidadas. O sea, en una celda podemos insertar una tabla con sus propiedades totalmente independientes de la tabla padre. Ésta es una de las funciones más poderosas que tienen los diseñadores web para lograr que sus páginas HTML se vean como ellos buscaban. Ahora veamos cómo quedará una tabla que nos muestra las ventas de una empresa de un cuatrimestre, divididas por rubro (**Figura 13**).

```

<html>
<head>
<title>Ventas del presente año </title>
</head>

```

## ▶ HTML COMPLETO

Todos los tags HTML se pueden anidar, y así es posible incluir texto en negrita e itálica, por ejemplo. Existen decenas de etiquetas adicionales. En [www.w3.org/TR/REC-html32](http://www.w3.org/TR/REC-html32) se puede acceder a la definición de todas las etiquetas compatibles con HTML 3.2.

```

<body bgcolor="#FFFFFF">
<p align="center"><font size="4">Ventas del presente a&ntilde;o</font></p>
<table width="99%" border="1" cellspacing="2" cellpadding="0"
  bordercolor="#3399FF">
  <tr>
    <td width="20%" bgcolor="#CCCCFF">&nbsp;</td>
    <td align="center" width="20%" bgcolor="#CCCCFF"><b>Enero</b></td>
    <td align="center" width="20%" bgcolor="#CCCCFF"><b>Febrero</b></td>
    <td align="center" width="20%" bgcolor="#CCCCFF"><b>Marzo</b></td>
    <td align="center" width="20%" bgcolor="#CCCCFF"><b>Abril</b></td>
  </tr>
  <tr>
    <td width="20%" bgcolor="#CCCCFF">Comestibles</td>
    <td width="20%" align="right">100</td>
    <td width="20%" align="right">200</td>
    <td width="20%" align="right">300</td>
    <td width="20%" align="right">150</td>
  </tr>
  <tr>
    <td width="20%" bgcolor="#CCCCFF">Electr&oacute;nicos</td>
    <td width="20%" align="right">1000</td>
    <td width="20%" align="right">1230</td>
    <td width="20%" align="right">880</td>
    <td width="20%" align="right">2349</td>
  </tr>
  <tr>
    <td width="20%" bgcolor="#CCCCFF">Limpieza</td>
    <td width="20%" align="right">78</td>
    <td width="20%" align="right">90</td>
    <td width="20%" align="right">78</td>
    <td width="20%" align="right">88</td>
  </tr>

```



## VERSIONES HTML

Existen distintas versiones de HTML disponibles, como la 3.2 y la 4.0. Cada una de ellas soporta diversas etiquetas y propiedades de cada una de ellas. Por eso debemos tener esto en cuenta a la hora de diseñar una página si queremos que sea cien por cien compatible con todos los navegadores del mercado, ya que algunos usuarios no podrán apreciar la página completa.

```

</tr>
<tr>
  <td width="20%" bgcolor="#CCCCFF">Vestimenta</td>
  <td width="20%" align="right">302</td>
  <td width="20%" align="right">a378</td>
  <td width="20%" align="right">290</td>
  <td width="20%" align="right">456</td>
</tr>
</table>
</body>
</html>

```

Ventas del presente año				
	Enero	Febrero	Marzo	Abril
Comestibles	100	200	300	150
Electrónicos	1000	1230	880	2349
Limpieza	78	90	78	88
Vestimenta	302	a378	290	456

**Figura 13.** Podemos ver el resultado de nuestra tabla de ventas en HTML. Mezclando propiedades, bordes y fondos, podemos lograr que su diseño sea más atractivo.

## Formularios

Como habíamos dicho con anterioridad, lo interesante de Internet es la interactividad y la posibilidad que tienen los usuarios de participar activamente del contenido de un sitio. HTML permite la interacción usuario-servidor mediante lo que llama

## ★ FILAS COLUMNAS O COLUMNAS FILAS

Al crear una tabla, por convención, primero se crean las filas y luego las columnas, o sea, no podrá existir una etiqueta `<tr>` dentro de una `<td>`. La única excepción es trabajar con tablas anidadas, pero en ese caso mediará una etiqueta `<table>`.

**formulario.** Un formulario es un área del HTML encerrada por el tag de apertura y cierre **<form>**, en donde hay una serie de controles que permiten el ingreso de información por parte del usuario que luego será enviada al servidor.

Dichos datos pueden ser textos introducidos por el usuario, una selección que realizó sobre una lista, una imagen o archivo que el usuario envía, o una casilla de selección. Es importante aclarar que escapa al lenguaje HTML la manipulación de dichos datos. Los datos se dirigen hacia otro archivo en el servidor (que puede ser procesado por ASP.NET), quien será el encargado de tomar decisiones sobre ellos, como, por ejemplo, enviarlos por e-mail, guardarlos en una base de datos o mostrar contenido personalizado en un HTML.

Una de las propiedades del tag **FORM** es **action**, que establece el archivo destino de los datos del formulario, cuando el usuario invoque el envío de datos. Cuando el usuario finalice, presionará la tecla **Enter** o pulsará un botón de tipo **Submit** que dirá **Enviar** o algo similar. En ese momento los datos que ha cargado viajarán al archivo establecido en **action**; asimismo, el navegador irá hacia esa dirección, como si fuera un link.

Otra propiedad muy importante de los formularios es **method**. Existen dos métodos de envío de información: **GET** y **POST**. Cuando enviemos información mediante el método **GET**, los datos se adjuntarán en la dirección URL de la página destino. Por ejemplo, si el **action** es **enviar.aspx** y en el formulario pedimos el nombre y apellido del usuario, la dirección destino será: **enviar.aspx?nombre=Maximiliano&apellido=Firtman**.

El signo de interrogación (?) separa el nombre del archivo de los valores. Luego de éste siempre se ubica el nombre del valor, un símbolo de igualdad (=) y el dato solicitado. Por cada dato adicional se deberá incorporar el separador **&** y **valor=dato**. Por otro lado, tenemos el método **POST**. Este método envía los datos en el momento de hacer la petición del archivo destino y éstos no son visibles por el usuario ni por el navegador. Así, la dirección destino será sólo **enviar.aspx** y los datos viajarán en la cabecera del protocolo HTTP sin que los veamos.



## TABLAS DE BASES DE DATOS

Podemos pensar que la tabla de ventas del ejemplo puede representar datos de una base de datos. Sean datos estáticos (HTML puro) o datos obtenidos de una base de datos (mediante ASP.NET, por ejemplo), el resultado hacia el navegador será exactamente el mismo: etiquetas de tabla, filas y columnas con texto y/u otras etiquetas dentro de cada celda.

¿Cuál utilizar, entonces? Ésta es una pregunta que también dependerá de la decisión del programador. Para ello, hay que tener en cuenta algunas cosas:

- Algunos navegadores no soportan más de 255 caracteres en la línea de direcciones (URL), por lo que si la información enviada por el método **GET** supera esa cantidad de caracteres, los datos serán truncados (borrados a partir de la posición 256), con la consecuente pérdida de información.
- El envío de archivos adjuntos se debe realizar obligatoriamente con el método **POST**.
- Cuando se trate de información vital (contraseñas, números de tarjeta de crédito, etc.) se debe utilizar **POST** para tener mayor seguridad.
- Las URLs trabajan con un encoding de 7 bits, lo que no permite el uso de caracteres especiales, como acentos, eñes, etc. Éstos serán convertidos a un código especial que luego debe ser reconvertido a la normalidad.
- En la URL tampoco deberían existir espacios en blanco, ya que no son soportados, por lo que si, por ejemplo, el nombre de una persona es **Juan Pablo**, al realizarse el envío, éste es transformado a **Juan%20Pablo**. El código **%20** es el que identifica al espacio en blanco.
- Cuando los datos por enviar son pocos (uno o dos), como un identificador, código o documento, es recomendable usar **GET**, ya que es más veloz.
- Si utilizamos el método **GET**, podremos hacer uso del envío de datos de una página a otra sin necesidad de emplear un formulario, sino a través de hipervínculos indicando los datos dentro de la URL.

De esta manera, dos posibles *tag forms* serían:

```
<form action="enviar.asp" method="GET">  
  
(...etiquetas de formulario ...)  
</form>
```

## { } ENVÍO DE UN FORMULARIO

La única forma provista por HTML de enviar un formulario es mediante un botón **Submit**. A través del lenguaje de cliente JavaScript podremos enviar el formulario ante otros eventos, como el cambio de un dato en un campo de texto o en una lista de selección.

```
<form action="enviar.asp" method="POST">
(...etiquetas de formulario ...)
</form>
```

En una misma página HTML puede haber más de un formulario, mientras uno esté fuera del otro (siempre debe estar cerrado el **form** anterior antes de abrir uno nuevo). Para identificarlos, también se puede utilizar la propiedad **name**, que permite nombrar a cada uno de ellos.

Ahora veamos los controles disponibles para incluir dentro de un formulario. Hasta ahora hemos escrito código, pero nada de ello es visual.

## Insertar textos

En Internet es muy común completar formularios con nuestros datos, por ejemplo al solicitar una cuenta de correo o al registrarse como usuario en una página. Es ahí cuando nos encontramos con casilleros de texto. Un casillero de texto es una zona en forma rectangular que permite el ingreso de datos (texto, números y símbolos) dentro de un formulario para ser enviado al servidor. La etiqueta de sólo apertura que administra este tipo de ingreso de datos es **input**. Por ejemplo:

```
<input type="text" name="nombre">
```

Otra posibilidad de ingresar texto es la etiqueta de apertura y cierre **textarea**. Este casillero de texto permite ingresar más de 255 caracteres distribuidos en varias líneas que el usuario podrá manipular mediante la tecla **Enter** y las flechas del cursor.

```
<textarea>Ingrese su descripción</textarea>
```

El texto que esté dentro de las etiquetas **<textarea>** y **</textarea>** es el valor predeterminado que el campo tendrá al iniciarse la página.

## ★ GET

Es importante saber que cuando utilizamos el método **GET**, todos los datos introducidos por el usuario serán visibles en la barra de direcciones del navegador. Esto puede traer problemas de seguridad cuando, por ejemplo, estamos enviando una contraseña y ésta no debería verse en pantalla.

## Listas o combos

Otra posibilidad de introducción de datos es el uso de listas o combos de selección. Esto permite al usuario seleccionar una opción de una lista predefinida por el programador. Esa opción será enviada al servidor. El tag de apertura y cierre que administra las listas es **select**.

Ahora bien, lo que todavía nos falta definir es el listado de ítem que existirán en la selección. Para ello utilizamos el tag de apertura y cierre **option**, que se debe encontrar dentro del tag **select**. Éste es un ejemplo de una lista:

```
<select name="pais">
  <option>Argentina</option>
  <option>México</option>
</select>
```

## Botones

Hasta ahora ya colocamos en nuestro formulario casilleros de introducción de texto y listas. Pero nos falta poder incluir botones para que el usuario presione al finalizar de escribir los datos. Un botón es un rectángulo de texto que al hacer clic con el mouse acciona el formulario.

Existen tres tipos de botones: **submit**, **reset** y botones de uso general. Estos últimos deben ser manipulados con otros lenguajes como JavaScript, por lo que no los analizaremos en este libro. El botón **submit** es el que hace accionar el formulario y enviar los datos al servidor, mientras que el botón **reset** se utiliza para limpiar todos los datos introducidos y empezar nuevamente como se encontraban al cargarse la página.

Siempre que tengamos que insertar botones de **submit** y **reset** en un formulario, debemos hacerlo en forma clara, para facilitarle al usuario la carga de datos en el formulario. A continuación veremos cómo establecer un botón **submit**:

## III STATE LESS

HTTP, el protocolo de comunicación entre un browser y un servidor web, es **state-less**, es decir, no mantiene el estado entre una petición y otra. La conexión se cierra al terminar la transferencia de una página, y por ello, en principio, no hay forma directa de relacionar la conexión constante de un usuario ni los valores existentes entre distintas páginas.

```
<input type="submit" name="boton" value="Enviar">
<input type="submit" name="boton" value="Enviar y Cerrar">
```

En general se utiliza un solo botón **submit** por formulario, pero, como en este ejemplo, también es posible incluir dos botones con dos valores distintos de **value** que accionen diferentes mecanismos una vez enviados los datos.

Un botón **reset** se realiza de la siguiente manera:

```
<input type="reset" value="Empezar de nuevo">
```

## Otros controles

Antes de finalizar con el tema de formularios, veremos cómo incluir un botón de radio, una casilla de verificación y campos ocultos.

Los botones de radio son una serie de valores entre los cuales el usuario puede seleccionar sólo uno. A diferencia de la lista de selección, los botones de radio son mostrados todos en pantalla sin necesidad de una barra de desplazamiento. Es muy útil para realizar preguntas con respuestas como “**Sí**” o “**No**” o **Multiple Choice**.

Veamos un ejemplo sencillo:

```
¿Es usted extranjero?<br>
<input type="radio" name="extranjero" value="si" checked>Sí<br>
<input type="radio" name="extranjero" value="no">No
```

Los casilleros de verificación permiten que el usuario seleccione o deseleccione una propiedad con un tilde. Esto es muy útil para que el usuario seleccione si le interesa una propiedad o no, sin necesidad de tipear él mismo la palabra.



## TEXTAREA

Los campos de tipo **textarea** permiten introducir textos largos como mensajes, reclamos, artículos, etc. Este tipo de dato se relaciona con los tipos de datos **Memo** o **Text** de bases de datos como Access o SQL Server, sin límite (en teoría) de cantidad de texto para incluir.

Veamos un ejemplo del código necesario para crear casilleros de verificación:

```
<input type="checkbox" name="interesado" value="si">Deseo  
  recibir información.
```

Y, por último, veremos cómo enviar datos definidos por el programador dentro de un formulario, sin la acción directa del usuario. Es el tipo de control **hidden** (oculto), que no es visible al usuario. Se define así:

```
<input type="hidden" name="variable" value="valor">
```

## Ejemplo de formulario

Luego de haber conocido las distintas opciones disponibles para crear un formulario en HTML, veamos un ejemplo terminado (**Figura 14**).

```
<html>  
<head>  
<title>Ejemplo de Formulario</title>  
</head>  
<body>  
<form action="enviar.aspx">  
  <p> Nombre:  
    <input name="nombre" type="text" maxlength="30">  
  </p>  
  <p> Apellido:  
    <input name="apellido" type="text" maxlength="50">  
  </p>  
  <p> Sexo:  
    <input name="sexo" type="radio" value="F" name="sexo">
```

### III VALUE

El valor que incluyamos en **value** será el texto que el usuario verá dentro del botón, por lo que es recomendable utilizar algún verbo en infinitivo, como **Enviar**, **Suscribir**, **Ir**, etc.

### ★ CUIDADO CON LOS RESET

Es recomendable que el valor del botón **Reset** sea claro, ya que es muy engorroso para el usuario equivocarse, presionar el botón y tener que ingresar todos los datos nuevamente.



**A no olvidarse:** no debemos olvidarnos, por nada del mundo, de incluir todos los componentes de un formulario dentro de una etiqueta `<form>` y `</form>`, para que su contenido pueda ser enviado correctamente al archivo ASP.NET que los recibirá y procesará.

Los ejemplos que vimos anteriormente sobre los controles disponibles para insertar en un formulario, se basan en un modelo de formulario común y corriente de los cientos de miles que encontramos en Internet. Desde ya que también se pueden crear formularios más complejos, que contengan diversas opciones, como la posibilidad de insertar imágenes, etc. Por ejemplo, también podemos utilizar el tipo de datos oculto, que a menudo se aplica para pasar datos de una página a otra mediante los formularios como claves o nombres de usuario, que ya fueron definidos con anterioridad y no se le piden al usuario nuevamente en este formulario. Ahora quedará en manos del lector explorar esas otras opciones.

## RESUMEN

En este capítulo describimos las diferentes tecnologías disponibles para trabajar en forma dinámica con HTML, además de explicar por qué HTML es un lenguaje estático por sí mismo. Repasamos la historia de los lenguajes de servidor y dimos una introducción al lenguaje HTML. Vimos que es un lenguaje para páginas web que se compone de etiquetas y propiedades, y que mezclándolas podremos lograr el diseño web requerido. Analizamos las tablas como uno de los objetos más interesantes de HTML, así como el uso de formularios para el envío de información que genera el usuario.



## TEST DE AUTOEVALUACIÓN

- 1 ¿Qué es un lenguaje dinámico?
- 2 ¿Qué cambios propuso la tecnología .NET?
- 3 ¿Cuáles son las características principales del lenguaje HTML?

```

<html><head>
<title>Yahoo!</title>
<meta http-equiv="P1C3-label" content="P1C3-1.1" http://www.iana.org/assignments/iana-1-rfc/rfc3280.html">
</head>
<body id="top" topmargin="0" margin="0">
<center>

</center>
<table border="0" style="width: 100%; text-align: center;">
<tr>
<td style="width: 50%; vertical-align: top;">
<input type="text" value="Buscar" style="width: 90%; height: 20px; border: 1px solid #ccc; margin-bottom: 5px;"/>
<input type="submit" value="Buscar" style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;"/>
</td>
<td style="width: 50%; vertical-align: top;">
<input type="text" value="http://www.yahoo.com/search" style="width: 90%; height: 20px; border: 1px solid #ccc; margin-bottom: 5px;"/>
<input type="submit" value="Buscar" style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;"/>
</td>
</tr>
</table>
</body>
</html>

```

- 4 ¿Qué es un formulario? ¿Qué elementos lo componen? ¿Cómo se crean?

**Ejemplo de Formulario** - Microsoft Internet Explorer

Nombre:

Apellido:

Sexo:  Femenino  Masculino

País de Origen: 

- Argentina
- México
- España

Deseo recibir información

Comentarios:

## EJERCICIOS PRÁCTICOS

- ✓ Es recomendable comenzar a analizar el código fuente de algunos sitios que queramos ver, utilizando Ver\Código Fuente desde nuestro navegador. Las etiquetas que no conozcamos podemos buscarlas en la Web para analizar su función y sus propiedades. Por último, sería recomendable buscar, dentro de páginas web, código dinámico de cliente e identificarlo.