ASP.NET. Introducción a las aplicaciones Web.

Creación de aplicaciones distribuídas.

Desde el momento en que la web se vislumbró como algo más que un vehículo para simples páginas estáticas de hipertexto (texto, imágenes e hiperenlaces) la posibilidad de ejecutar código tanto en el cliente como en el o los servidores se convirtió en una prioridad.

Pronto se vio que la ejecución de código en el cliente (scripts de cliente, `applets`, controles ActiveX) servía para temas gráficos y validación de datos de formularios pero no era soportado del mismo modo por todos los clientes. Si además se deseaba utilizar para otros fines, como acceso a datos almacenados en el servidor, presentaba problemas de seguridad y de carga excesiva de tráfico en la red.

Por otro lado, la ejecución de código en el servidor, invocado desde el cliente, eliminaba estos problemas. El cliente pedía sólo la información deseada mediante una petición HTTP y el servidor Web invocaba la ejecución de un código que generaba esa información al momento (dinámicamente) y se la enviaba al cliente. Para generar dicha información, el código de servidor solía acceder a bases de datos (o ficheros, etc...) que no tenían porqué residir en el mismo equipo físico que el servidor Web y el código invocado por éste. Este hecho provocó que una arquitectura típica de aplicaciones Web fuese la llamada arquitectura de tres capas:

- *Capa cliente*: Máquina cliente, con un navegador o aplicación similar.
- *Capa intermedia*: Servidor Web más código invocado por éste, que es el que se encargaba de generar la respuesta a enviar al cliente (generalmente HTML). Al código invocado se le suele llamar "lógica de negocio" porqué es el que realmente genera lo que espera el cliente, manipulando si es necesario los datos del servidor de datos.
- *Capa de datos*: Normalmente es otro sistema físico, separado de la capa intermedia, en el que está instalado el SGBD que provee los datos. La lógica de negocio accede a esta capa bien a través de Internet, de una intranet u otro tipo de red, no teniendo porque ser HTTP el protocolo utilizado.

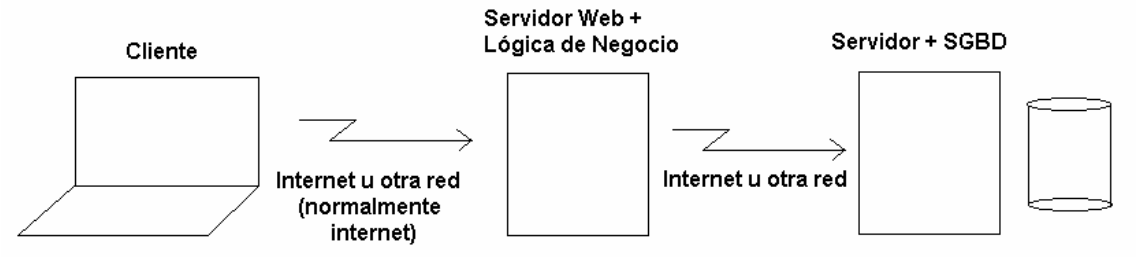

Figura 20.1. Arquitectura de tres capas.

Si bien la arquitectura de tres capas era y es muy utilizada. En ocasiones se consideran también arquitecturas de dos a n capas. La arquitectura de dos capas es la llamada

arquitectura cliente-servidor. En ésta tanto la lógica de negocio como el SGBD residen en la misma capa o equipo.

La arquitectura de n capas (n>3) consiste en distribuir la lógica de negocio -y en algunos casos la fuente de datos, sea un SGBD u otra fuente- en varias capas o equipos que se comunican entre sí a través de una red, que generalmente es Internet o una intranet tipo Web.

A las aplicaciones de dos o más capas se les llama aplicaciones distribuidas porque las distintas partes de aplicación (cliente-interfaz, lógica de negocio-cerebro, datos-memoria) están físicamente distribuidas.

Las primeras aplicaciones Web distribuidas permitían al cliente invocar la ejecución de un programa en el equipo con el servidor Web utilizando la pasarela CGI (*Common Gateway Interface*). La pasarela CGI es un módulo que puede ser utilizado por el servidor Web para lanzar un ejecutable, pasarle los parámetros de una petición de un cliente -en la que se incluye el nombre del ejecutable- y recoger los resultados del ejecutable para devolvérselos al cliente. Obviamente, si el cliente es un navegador Web, esperará que el ejecutable invocado devuelva texto HTML. Por supuesto, el ejecutable puede acceder a una base de datos, realizar cálculos, etc...

La pasarela CGI presentaba problemas de rendimiento -lanzaba un proceso por petición de cliente- y aunque esto se fue mejorando con soluciones como fastCGI, se dio prioridad a la búsqueda de nuevos modos de ejecutar código en las capas intermedias -y final- de una aplicación distribuida.

Microsoft incorporó a su servidor IIS la posibilidad de recibir peticiones de cliente que invocasen lo que se dio en llamar filtros y extensiones ISAPI. La idea era similar a la invocación de ejecutables mediante la pasarela CGI pero en este caso no era necesaria la pasarela ni se creaba un proceso nuevo por cada petición de cliente. Se creaba un hilo de ejecución que es mucho más económico que un proceso, sobre todo en Windows. El problema de ISAPI era que la programación de filtros y extensiones ISAPI requería conocimiento de programación sobre Windows medio-alto.

De este modo, el reto se convirtió en ofrecer una solución de computación distribuida no basada en CGI pero sencilla a la vez. En este caso tanto Microsoft (ASP) como Netscape-SUN (JSP) respondieron con una solución similar. Esta solución consistía en incluir la lógica de negocio en las propias páginas Web. Así, las nuevas páginas consistían en una mezcla entre texto HTML y scripts de código de servidor. Cuando se solicitaba una página de este tipo, el servidor la iba leyendo y devolviendo al cliente el texto HTML tal y como estaba, pero cuando llegaba a un script o trozo de código de servidor, lo compilaba (al principio se interpretaba en lugar de compilar) y ejecutaba, devolviendo al cliente el resultado de ejecutar tal código (que podía ser el resultado de una consulta a una base de datos o de un cálculo complejo,...). El cliente obtenía generalmente una página HTML -posteriormente aparecerán nuevos lenguajes de marcas- de la que parte venía tal y como era en el servidor y parte había sido generada dinámicamente al ejecutar el script que ocupaba originalmente su lugar.

A la solución planteada por Microsoft se le llamó *Active Server Pages* (ASP) y consistía en páginas HTML con scripts en lenguaje VBScript (un subconjunto de Visual Basic) o

JScript (un subconjunto de Java). A la solución planteada por Netscape-SUN se la llamó *Java Server Pages* (JSP) y consistía en páginas HTML con scripts en lenguaje JavaScript (un subconjunto de Java).

La ventaja del código de un script es que es sencillo (versiones recortadas de Java o Visual Basic) y no hace falta compilarlo, ya que se compila al ser invocado (lo cual, por otro lado lo hace más lento). No obstante, un script, en muchos casos se queda corto. Para solucionar este problema se permitió poder invocar la ejecución de programas desde un script (componentes, para ser exactos).

Desde un script ASP se podían y pueden utilizar controles ActiveX y desde un script JSP se pueden utilizar servlets.

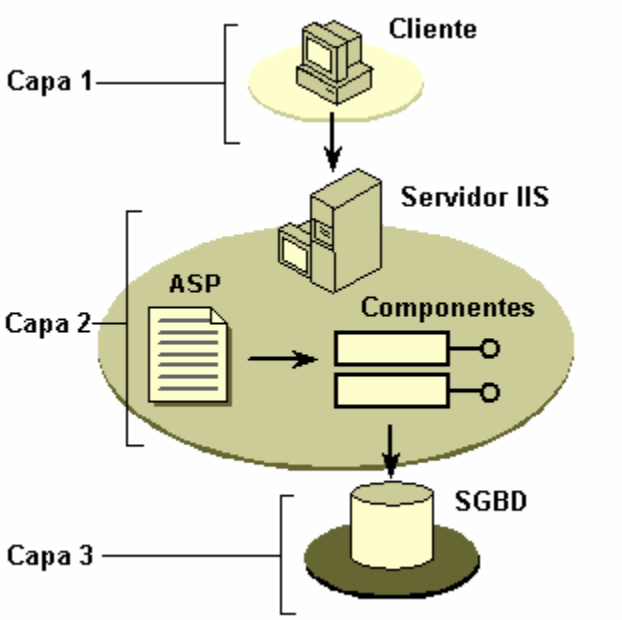

Figura 20.2. Arquitectura de tres capas con la tecnología ASP.

Con la tecnología .NET, en concreto con ASP.NET se ha buscado limar los problemas que planteaba ASP.

Comparativa ASP y ASP.NET.

Tanto ASP como ASP.NET permiten la generación dinámica de contenido, que generalmente es fruto del procesamiento de unos datos. Aunque la idea es la misma, ASP.NET mejora ciertos puntos de ASP.

Posiblemente, la mejora de mayor importancia es que los scripts en ASP.NET sólo son compilados una vez, la primera que son invocados. A partir de este momento la `dll` con el script compilado es almacenada (carpeta `codegen`) e invocada cada vez que se solicita el script. Esta manera de trabajar hace que la ejecución sea más rápida que si cada vez que se solicita el script hubiera que compilarlo o interpretarlo, que es el funcionamiento de ASP.

Por otro lado, ASP.NET admite cualquier lenguaje compatible .NET para los scripts (C#, Visual Basic .NET, JScript...). En cambio, ASP sólo admite VBScript o JScript.

Otro punto importante es cuando se desea invocar código ya existente desde un script. ASP permite la invocación de componentes COM/DCOM (ActiveX es un subconjunto de éstos), lo cual ofrece una gran potencia a los scripts pero presenta el problema de que es necesario registrar -en el registro del sistema- los componentes que van a ser utilizados (`regsvr32.exe`). Cualquier cambio que se realice sobre el componente obligará a parar su uso y volverlo a registrar para que pueda ser utilizado de nuevo.

En el caso de ASP.NET se permite invocar componentes .NET y el registro es innecesario, basta con copiar el componente a la carpeta deseada e invocarlo. Además, si se modifica el componente, no es necesario dejar de utilizarlo para sustituirlo, puede sobrescribirse directamente. Esto es así porque no se bloquea la librería del componente (se trabaja con una copia y con la memoria).

WebForms

ASP.NET permite crear aplicaciones distribuidas en las que la parte de la aplicación correspondiente a la interfaz de usuario se puede desarrollar mediante WebForms (`namespace System.Web`) y la parte de servidor se puede desarrollar codificando directamente los métodos de respuesta a los eventos de la interfaz WebForms o bien mediante WebServices (`namespace System.Web.Services`).

De este modo, WebForms es una parte de la tecnología ASP.NET, que permite crear interfaces de usuario para aplicaciones Web. Estas interfaces de usuario son independientes de la aplicación cliente (navegador) utilizada para representarlas, debido a que toda la parte de proceso que hay en éstas se ejecuta en el servidor.
Es posible utilizar WinForms en lugar de WebForms para crear la parte de usuario de una aplicación Web. En este caso, la parte de usuario sólo podrá ejecutarse en una máquina que tenga el CLR (no como un WebForm, que es soportado por cualquier navegador).

Los WebServices son componentes que se ejecutan en el servidor y generalmente incluyen la lógica de negocio. Al igual que los componentes tradicionales, encapsulan una funcionalidad específica y pueden ser llamados desde diferentes programas. A pesar de que se utilicen en el servidor, los WebServices son accesibles a través de protocolos Web (HTTP,...), lo cual los hace compatibles con aplicaciones creadas en diferentes lenguajes, que se ejecuten en máquinas diferentes y con sistemas operativos distintos.

Un punto importante de los WebServices es que, en aplicaciones Web que soporten una carga alta de peticiones de cliente, pueden utilizarse para balancear dicha carga y bajarla, registrando el mismo servicio en varios puertos y haciendo que los clientes los llamen directamente, saltándose el cuello de botella que puede ser una única página Web de acceso al servicio por la que tengan que pasar todas las peticiones.

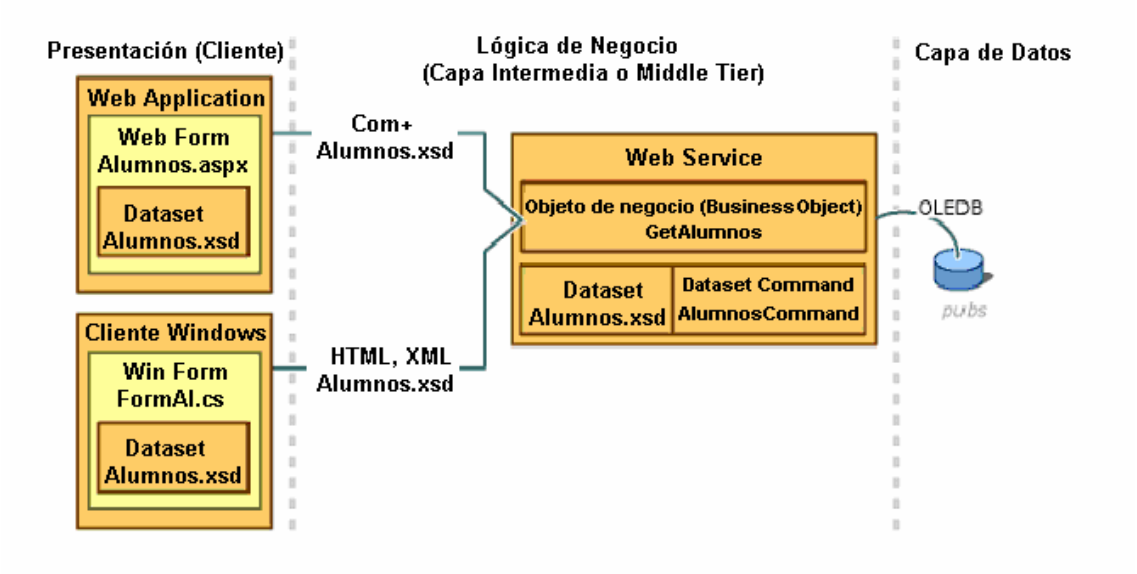

Figura 20.3. Arquitectura de 3 capas con WebForms y con WebServices.

Características de los WebForms.

Las principales características de los WebForms son:

- WebForms es una parte de la tecnología ASP.NET, en concreto son un modelo de programación que puede ser utilizado en el servidor para crear dinámicamente páginas Web.

- Pueden ser programados utilizando cualquier lenguaje del Framework .NET, como Visual Basic, C#, Managed extensions for C++, JScript.NET...

- Están construidos sobre el Framework .NET y proveen de todos los beneficios de estas tecnologías, incluida la ejecución en un entorno controlado, de tipos seguros y con herencia.

- Son soportados por Visual Studio, que ofrece herramientas RAD (*Rapid Development*) para diseñar y programar los Formularios Web (WebForms).

- Soportan un amplio conjunto de controles basados en el servidor que ofrecen la posibilidad de desarrollar formularios estándar en un entorno de diseño RAD.

- Pueden ser extendidos mediante controles creados por el desarrollador y por otras empresas.

- Las aplicaciones son independientes del navegador: WebForms ofrece un framework o marco para crear la lógica de las aplicaciones en el servidor devolviendo al cliente texto HTML o XML, lo cual elimina la necesidad de codificar explícitamente para los diferentes navegadores. No obstante, los navegadores Microsoft, como el Internet Explorer 5 pueden aprovechar las características de un cliente Web rico.

- WebForms ofrece un modelo de programación basado en eventos: WebForms permite implementar el mecanismo de eventos en las aplicaciones Web de un modo transparente. El Framework .NET abstrae el modelo de eventos de modo que el mecanismo de captura de un evento en el cliente, la transmisión al servidor y la llamada al método adecuado es automática y transparente.

- El modelo de objetos es abstracto e intuitivo: El Framework .NET para WebForms ofrece un modelo de objetos que permite pensar en los Forms como en una unidad, en lugar de dos partes separadas de cliente y servidor. Los controles WebForms de servidor son una abstracción del contenido físico de una página HTML y de una interacción directa entre el navegador y el servidor. De este modo es posible utilizar controles WebForms de servidor del mismo modo que se utilizarían si fuesen de cliente y no es necesario preocuparse de crear el código HTML para presentar y procesar los controles y su contenido.

- Gestión de Estado: El Framework WebForms gestiona de modo automático el almacenamiento de la información de estado del Form y de sus controles, y todo ello sin sobreutilizar los recursos.

- Funcionamiento de servidor escalable: El Framework WebForms permite escalar una aplicación desde un ordenador con un sólo procesador hasta convertirla en una aplicación que funcione sobre un WebForm con múltiples ordenadores sin realizar cambios complicados a la lógica de la aplicación.

Conceptos básicos.

Un WebForm consiste en dos componentes, los elementos visuales y el código.
En Visual Studio, por defecto cada uno de esos elementos se almacena en un fichero separado. Los elementos visuales se crean en un fichero `.aspx` que se comporta como contenedor de los elementos HTML y los controles WebForms. El código se guarda en un fichero separado; si es C#, la extensión del fichero será `.cs`.

No obstante es posible mantener los elementos visuales y el código en el mismo fichero: el diseñador de WebForms del Visual Studio no lo hace por defecto.

La clase Page.

Aunque los WebForms se programen en módulos separados (por defecto), forman una unidad. Cuando se compila un WebForm, ASP.NET analiza la página y su código, genera dinámicamente una nueva clase y la compila. Esta clase deriva de la clase `Page` de ASP.NET, extendiéndola con el código añadido, con controles y con el texto estático HTML que hay en el fichero con extensión `.aspx`.

La clase derivada de `Page` se guarda en un fichero ejecutable que se ejecuta en el servidor cada vez que se pida la página. En tiempo de ejecución, la clase `Page` procesa las peticiones que llegan y responde enviando HTML generado dinámicamente al navegador del cliente. Si la página tiene controles de servidor, que es lo más común, la clase derivada de `Page` se comporta como contenedor de los controles y crea y envía al cliente el código HTML correspondiente a los controles.

Para los desarrolladores acostumbrados a ASP, este modelo de desarrollo y ejecución representa una novedad. Un procesador ASP lee una página ASP, que consiste en texto HTML y código mezclados, extrae y ejecuta (interpreta, no compila) únicamente el código y luego devuelve al cliente el texto HTML tal y como estaba y en donde estaba el código devuelve el resultado de ejecutarlo, que será texto HTML.

Con la clase `Page` sucede al revés. Todo el WebForm es un ejecutable cuya salida es de tipo HTML. Cuando se pide la página, ésta es ejecutada y sus métodos invocados. La página atraviesa un proceso en el que las etapas son análogas a las que atraviesan otros componentes (inicialización, proceso, eliminación) pero con algunas diferencias.

Por ejemplo: al igual que un componente Web, la clase `Page` atraviesa las etapas comentadas siempre que es llamada, es decir, la página es inicializada, procesada y liberada siempre que es pedida al servidor. Realmente, la información necesaria para reconstruir la página es cacheada, pero esto es independiente del ciclo de vida de la clase.

Otra diferencia es que en su ciclo de vida, la clase `Page` tiene etapas diferentes a las del ciclo de vida de un componente común, como pueden ser la etapa de generación de la respuesta HTML al cliente (a la que se llama `render`).

Derivación de la clase Page.

Cuando se desarrollan WebForms con Visual Studio, cada WebForm se compone por defecto de un fichero con la página Web (`.aspx`) y otro con la clase (`.cs`, si se está desarrollando en C#).

Por ejemplo, si se crea un proyecto de tipo WebForm con un Form al que se llama `WebPage1`, entonces se creará una clase `WebPage1` derivada de `Page`. La página (el fichero `.aspx`) es derivada de la clase `WebPage1`.

Debido a que un fichero `.aspx` no es un módulo desde el punto de vista tradicional, su relación con el fichero de la clase se establece mediante directivas situadas al principio de la página. En concreto, se utiliza la directiva `CodeBehind` para especificar el fichero que contiene la clase de la cual deriva el fichero `.aspx`. y la directiva `Inherits` para indicar la clase (`namespace.clase`). Un ejemplo de utilización de esta directiva puede ser:

```
<%@ Page language="c#" Codebehind="WebForm1.cs"
AutoEventWireup="false" Inherits="HolaInternet.WebForm1" %>
```

En Visual Studio, esta relación es mantenida automáticamente, incluso aunque se cambie de nombre el WebForm.

Componentes de los WebForms.

Como se ha comentado, los WebForms dividen las aplicaciones Web en dos partes:

- Los componentes visuales.

- La lógica de interfaz de usuario.

Para quien haya desarrollado aplicaciones con herramientas visuales, esta distinción es algo común.

Los componentes visuales van dentro del fichero `.aspx` (al que se llama página), que contiene código HTML pero también elementos específicos de tipo WebForm (Visual Studio permite añadirlos de un modo gráfico).

La lógica de la interfaz de usuario para el WebForm consiste en el código que el desarrollador crea para interactuar con el Form (como se ha dicho, esta lógica se guarda en un fichero de código, `.cs`). Al ejecutar el Form, la lógica se ejecuta y genera una respuesta de tipo HTML.

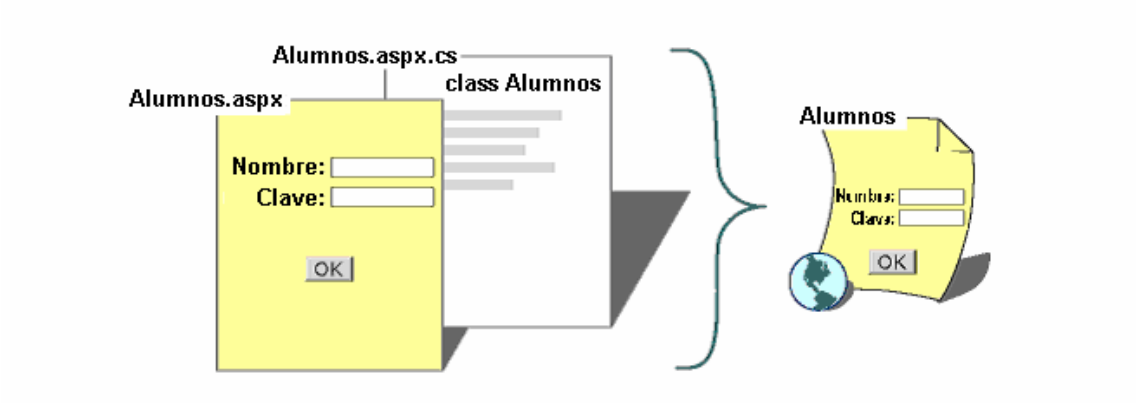

Figura 20.4. Un Web Form es el resultado de la unión entre los componentes visuales de la interfaz de usuario (`Alumnos.aspx`) y el código asociado a la interfaz (`Alumnos.aspx.cs`)

Al código asociado a la interfaz se le llama también *lógica de la interfaz de usuario*.

Es importante tener en cuenta que la lógica de la interfaz de usuario no tiene porqué incluir a la lógica de negocio. La lógica de interfaz de usuario tiene como principal cometido ser la lógica (el código) que gestiona el comportamiento gráfico (es un código que se introduce en los métodos de respuesta a eventos gráficos). La lógica de negocio manipula información y genera una respuesta (normalmente HTML o XML) para el cliente. Esta respuesta va mucho más allá de una simple gestión gráfica y suele ser llevada a cabo por WebServices. El caso es que el código correspondiente a un WebService puede introducirse en los métodos de respuesta a los eventos gráficos de un WebForm, es decir, donde va la lógica de la interfaz de usuario (no es la mejor opción pero puede hacerse, de hecho puede ser más justificado hacerlo a la inversa).

Problemas que resuelven los WebForms.

La programación de aplicaciones Web presenta problemas que no suelen presentarse cuando se programan aplicaciones tradicionales basadas en cliente. WebForms ayuda a superarlos. Algunos de estos problemas son:

- Creación de una interfaz de usuario rica: La creación de una interfaz de usuario con gran cantidad de contenido, diseño complejo y una gran interacción con el usuario, puede ser duro y tedioso utilizando HTML. Más aún teniendo en cuenta que no todos los navegadores soportan las mismas características.

- Separación cliente-servidor: En una aplicación Web, el cliente y el servidor son programas diferentes que, usualmente, se ejecutan en diferentes máquinas e incluso sobre sistemas operativos diferentes. Debido a esto, las dos partes de una aplicación Web comparten poca información y tienen una estructura simple.

- Ejecución sin registrar información: Cuando un servidor Web recibe una petición de una página, la encuentra, la procesa y la envía al navegador del cliente, descartando tras esto su información. De este modo, si el usuario vuelve a pedir de nuevo la misma página, el servidor repite toda la secuencia. El hecho de que el servidor no tenga memoria de las páginas que ha procesado hace que si se desea mantener información sobre una página deba hacerse mediante código.

- Desconocimiento de las características del cliente: Es complicado crear una aplicación que presente el mismo resultado a clientes diferentes y cuyas características, en muchos casos no son conocidas de antemano.

- Acceso a datos: Acceder a fuentes de datos en aplicaciones Web tradicionales puede ser complicado y consumir muchos recursos.

Diferencias y Similitudes entre los WebForms y los WinForms.

Cuando se diseñan aplicaciones con interfaz de usuario existen dos opciones: Windows Forms (también llamados WinForms) y WebForms.

Si se está desarrollando un *site* de comercio electrónico que será accedido desde cualquier cliente de Internet, lo lógico es diseñar la aplicación utilizando WebForms.
Si, en cambio, se está desarrollando una aplicación que implica mucho procesamiento y rapidez de respuesta, aprovechando al máximo los recursos del cliente, lo lógico será diseñar la aplicación utilizando WinForms.

No obstante, existen casos en los que no es tan claro el tipo de solución a elegir.

WinForms

Los WinForms se utilizan para desarrollar aplicaciones en las que se espera que sea el cliente el que lleve la carga de proceso. Estas incluyen las aplicaciones de escritorio de Win32. Ejemplos de ello son las aplicaciones de dibujo o manejo de gráficos, los sistemas de entrada de datos, los sistemas punto de venta y los juegos.

Debido a que una aplicación Windows basada en WinForms se diseña sobre el Framework de Windows, ésta tiene acceso a los recursos del sistema del ordenador cliente, como los ficheros locales, el registro de Windows, la impresora...

Las aplicaciones basadas en WinForms pueden hacer uso de GDI+, que ofrece los servicios gráficos de la plataforma .NET (es la librería gráfica).

WebForms

Los WebForms se utilizan para crear aplicaciones en las que el interfaz de usuario primario es un navegador
Las aplicaciones basadas en WebForms son independientes de la plataforma, no obstante pueden optimizarse para unos u otros navegadores (por ejemplo, para IE 5.0, 5.5 o 6.0). En muchos casos, las optimizaciones están en los componentes utilizados en los WebForms, que son capaces de detectar el tipo de navegador.
Siguiendo las ideas comentadas se puede concluir que las aplicaciones basadas en WebForms son adecuadas cuando se desea que los clientes tengan poco procesamiento, acceso limitado a ciertas partes de la aplicación y puedan ser heterogéneos. Además, al utilizar HTML y/o XML, las aplicaciones basadas en WebForms son interesantes también para tratamientos de texto en los que el formato sea importante.

Comparativa entre los WebForms y los WinForms.

Criterio/Característica	WinForms	WebForms
Instalación	La tecnología WinForms permite instalación automática, donde las aplicaciones pueden ser descargadas, instaladas y ejecutadas directamente en la máquina del usuario sin modificar el Registro del Sistema.	La tecnología WebForms no tiene instalación de clientes, basta con que el cliente disponga de un navegador. El servidor debe ejecutarse sobre el Framework .NET. Las actualizaciones de la aplicación se realizan actualizando el código en el servidor.
Gráficos	WinForms incluye GDI+, que permite utilizar gráficos sofisticados para juegos y otros entornos ricos en gráficos.	Es posible utilizar GDI+ en el servidor para crear gráficos y también otros sistemas, pero siempre han de ser creados en el servidor.
Capacidad de respuesta.	Los WinForms pueden ejecutarse completamente en la máquina del cliente, ofreciendo una rápida capacidad de respuesta.	Si los usuarios tienen el IE 5 o mayor, una aplicación WebForms puede aprovechar las capacidades DHTML para crear una interfaz de usuario rica y de respuesta rápida. Si los usuarios tienen otros navegadores habrá que cargar trabajo extra en el servidor.
Control de los formularios y del flujo de texto.	Los WinForms permiten posicionar los controles de un modo preciso mediante coordenadas (x, y). Para mostrar texto, lo normal es	Los WebForms se basan en el estilo HTML. Tienen una capacidad alta de formateo de texto, pero el posicionamiento de controles

	asignarlo a los controles, para lo cual existen propiedades. El formato es limitado.	está más limitado (si el navegador del cliente soporta DHTML, es posible realizar un diseño mucho más preciso, con coordenadas).
Plataforma	Los WinForms requieren el Framework .NET ejecutándose en el ordenador del cliente.	Los WebForms sólo requieren un navegador. Si el navegador soporta DHTML, ofrecerá características ventajosas. El servidor Web debe ejecutarse sobre el `Framework` .NET.
Acceso a los recursos locales (sistema de ficheros, Registro de Windows...).	La aplicación tiene acceso total a los recursos del ordenador. Si es necesario, puede restringirse el acceso a ciertos recursos.	La seguridad del navegador no permite acceder a los recursos del ordenador cliente.
Modelo de programación.	Los WinForms están basados en el lado del cliente, en el modelo de mensajes del cliente	Los WebForms se basan en un modelo sin conexión, asíncrono. Los componentes de la aplicación son invocados mediante HTTP. Esto hace que aplicaciones que requieran respuesta rápida o acceso concurrente de alto nivel a bases de datos no sean adecuadas para ser desarrolladas con WebForms.

Tabla 20.1

Creación de una página WebForms.

Cuando se desarrolla una página de tipo WebForms se pueden realizar los siguientes pasos.

- Crear una nueva página WebForms vacía.
- Añadir texto HTML y establecer sus atributos mediante la **ventana de propiedades**.
- Añadir controles arrastrándolos desde la **Caja de Herramientas** y estableciendo sus propiedades.
- Si se desea, convertir un control (o varios) HTML para que se ejecute en el servidor como un control WebForm.
- Añadir código a los controles (que se ejecutará en el servidor).
- Añadir manejadores de eventos a los controles.
- Ejecutar la página WebForm.

Creación de una página Web y adición de código HTML.

- Elegir **Archivo/Nuevo/Proyecto**, con lo que aparece el cuadro de diálogo **Nuevo Proyecto**.

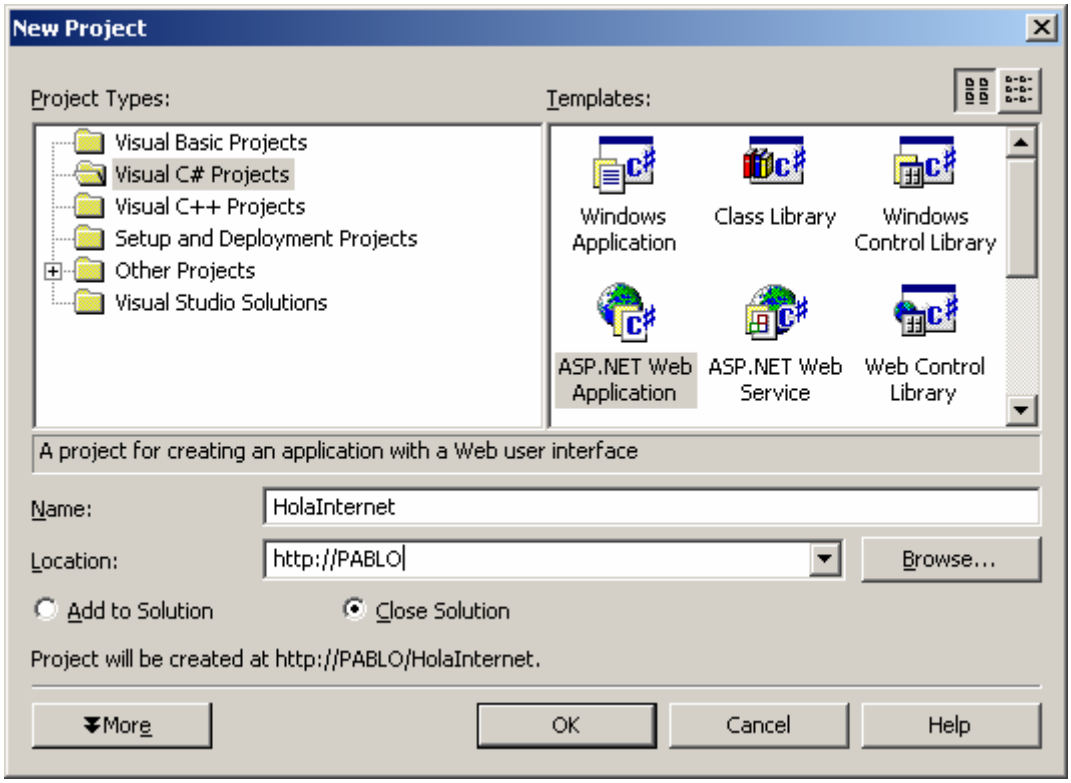

Figura 20.5. Ventana de creación de una nueva aplicación Web ASP.NET.

- Como se puede ver aparece un nuevo campo, **Ubicación**, en el que se ha de indicar la ruta al servidor Web donde se va a crear el proyecto. Tal servidor Web ha de ser el IIS de la versión 4 o superior y tener el Framework.NET debajo. Si el servidor no es correcto, se indica.

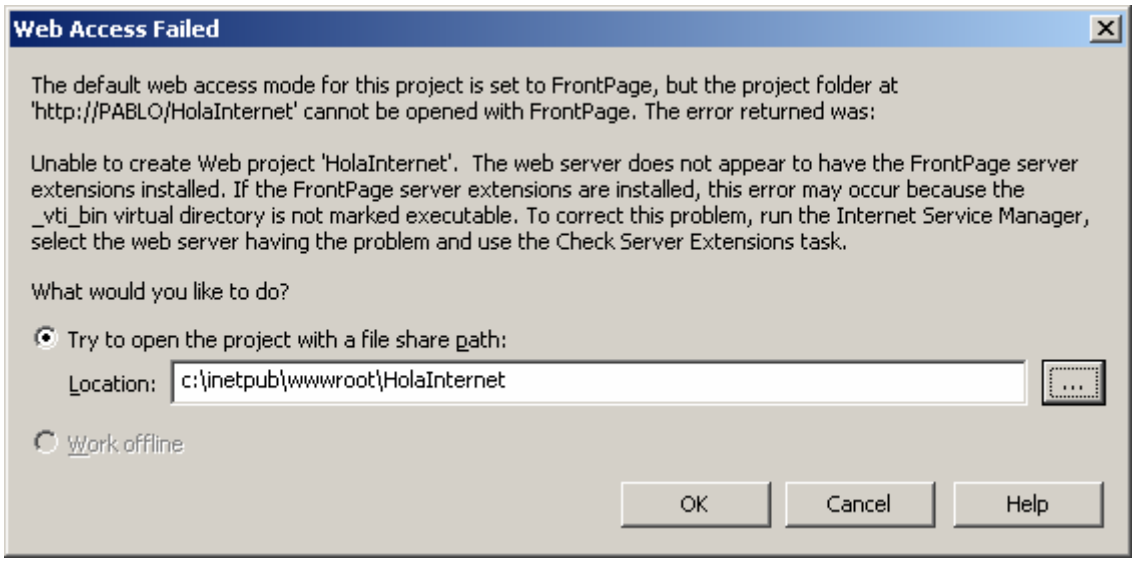

Figura 20.6. Cuadro de diálogo informativo de error en búsqueda de servidor Web del que "colgar" la aplicación.

- Si todo va bien se crea la aplicación `HolaInternet`, la cual contendrá una página llamada `WebForm1.aspx`

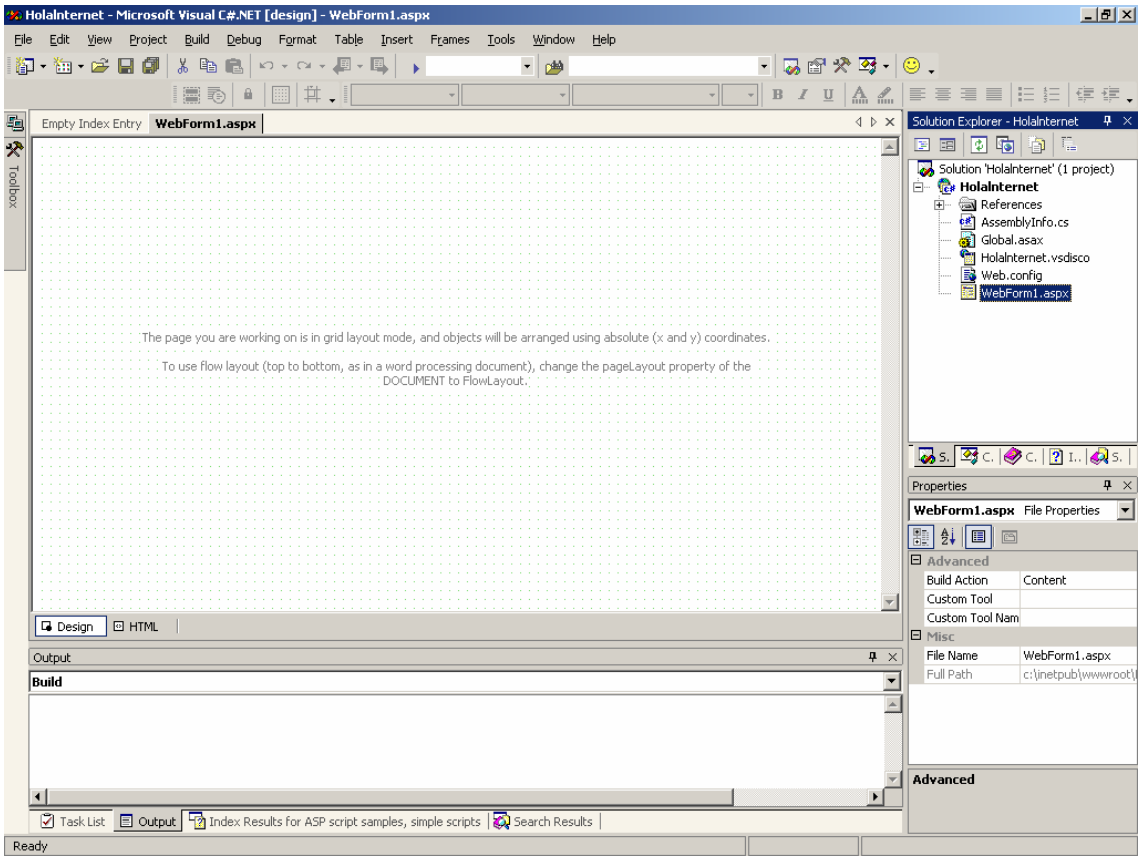

Figura 20.7. Aplicación Web en blanco creada por el asistente de Visual Studio .NET

Además de esa página se crean:

- Una solución llamada `HolaInternet`, que contiene la aplicación original.
- `AssemblyInfo.cs`, que contiene la descripción del `assembly`.
- `Global.asax`, que contiene información global y sobre eventos relacionados con los objetos aplicación y sesión web.
- `HolaInternet.disco`, que es un fichero que describe los WebServices que haya en el proyecto, permitiendo además el descubrimiento dinámico.
- `Web.config`: este fichero contiene información de configuración de la aplicación.
- `WebForm1.aspx`: es la primera página ASP.NET en la aplicación.

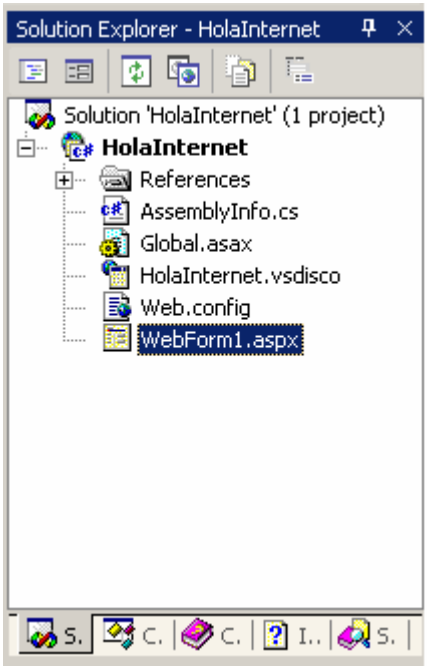

Figura 20.8. Relación de los ficheros más relevantes de la aplicación ASP.NET

- Una vez se tiene la página de servidor es posible utilizar el diseñador de Visual Studio .NET para diseñar la página. Un ejemplo posible puede ser:

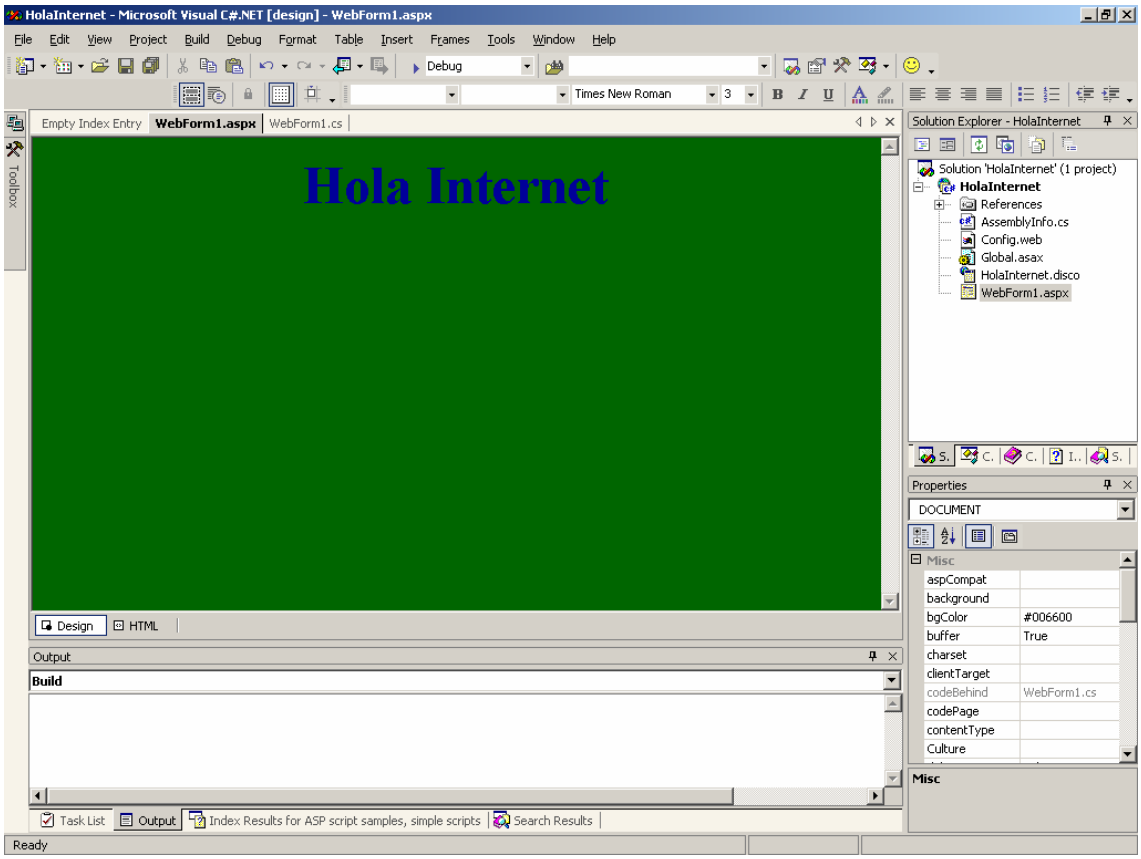

Figura 20.9. WebForm de la aplicación. Vista de diseño.

- Donde el código HTML es:

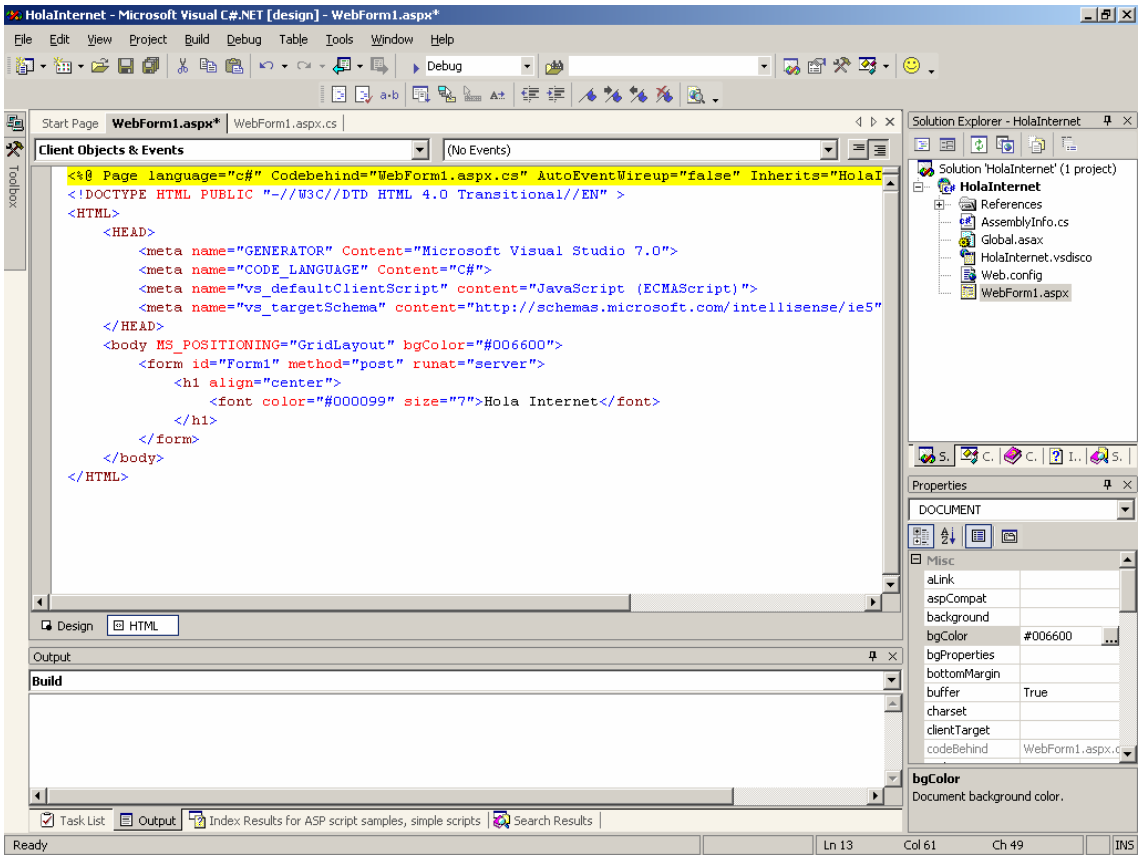

Figura 20.10. WebForm de la aplicación. Vista HTML.

Nota: Para cambiar el color de fondo se ha accedido el objeto DOCUMENT en la **ventana propiedades**. La adición del texto `Hola Internet` puede hacerse directamente desde la vista HTML, se puede comprobar de este modo la ayuda contextual que se muestra según se escribe.

El resultado de ejecutar esta página es:

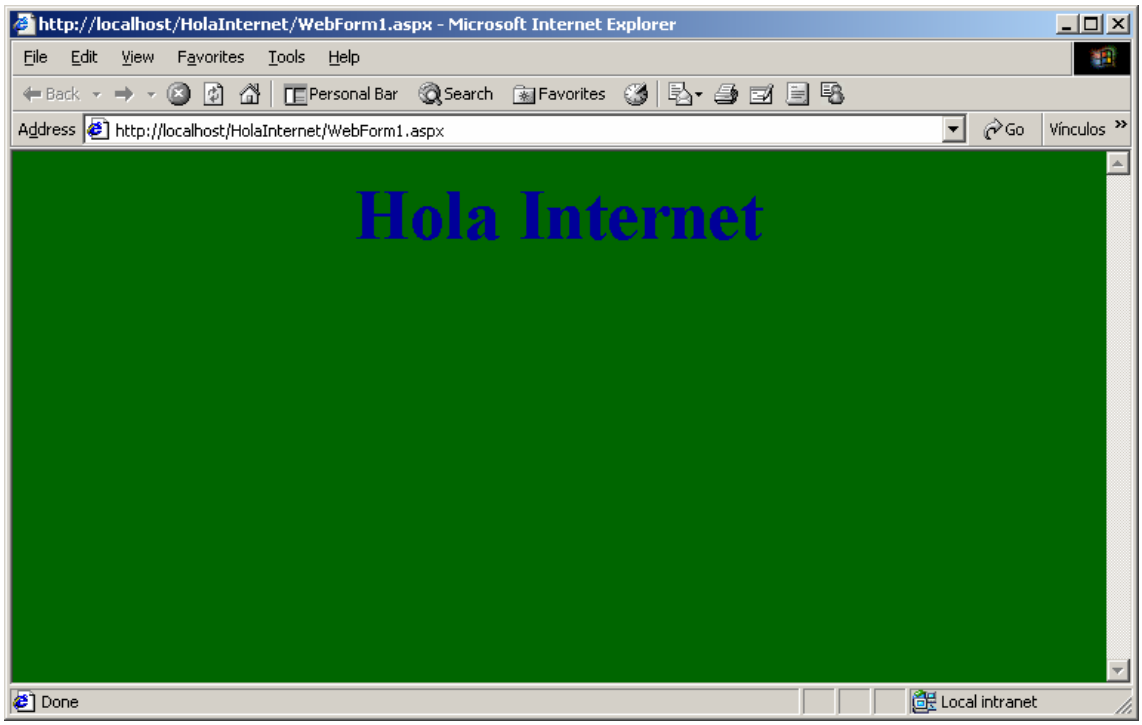

Figura 20.11. Resultado de ejecutar la aplicación (equivale a invocar el WebForm desde el Internet Explorer).

Adición de controles.

Es posible utilizar cuatro tipos de controles en una página Web:

- *Controles HTML de Servidor:* Cuando se añade un control HTML a un WebForm, por defecto éste es un control HTML típico de cliente. No puede asociársele código en el servidor. Si se modifica un control HTML típico añadiéndole la marca "`runat = server`" será considerado de servidor, tendrá una clase y eventos y código asociado que se ejecutará en el servidor.
- *Controles de servidor ASP.NET:* Son controles específicos WebForms que ofrecen más características que los controles HTML de servidor y no se asocian directamente a elementos HTML.
- *Controles de validación:* Son controles diseñados para validar de un modo sencillo los datos que introduzca el usuario.
- *Controles definidos por el desarrollador:* Estos controles pueden crearse en base a agrupaciones de otros controles e incluso a partir de una página ASP.NET.

Para añadir controles HTML a un WebForm se ha de utilizar la opción HTML de la **caja de herramientas**. De modo similar, para añadir a un WebForm controles ASP.NET de servidor, de validación o definidos por el desarrollador, se ha de utilizar la opción WebForms de la **caja de herramientas** (los de desarrollador han de añadirse antes a la **caja de herramientas** para que aparezcan).

Los controles ASP.NET de servidor cubren todo lo que pueden ofrecer los controles HTML y más. Se explicará cómo utilizar controles HTML pero no es realmente necesario.

Como ejemplo, se va a añadir una caja de texto y un botón al WebForm:

- La caja de texto como control HTML.
- El Botón como control ASP.NET.

Para añadir la caja de texto se realizan los siguientes pasos.

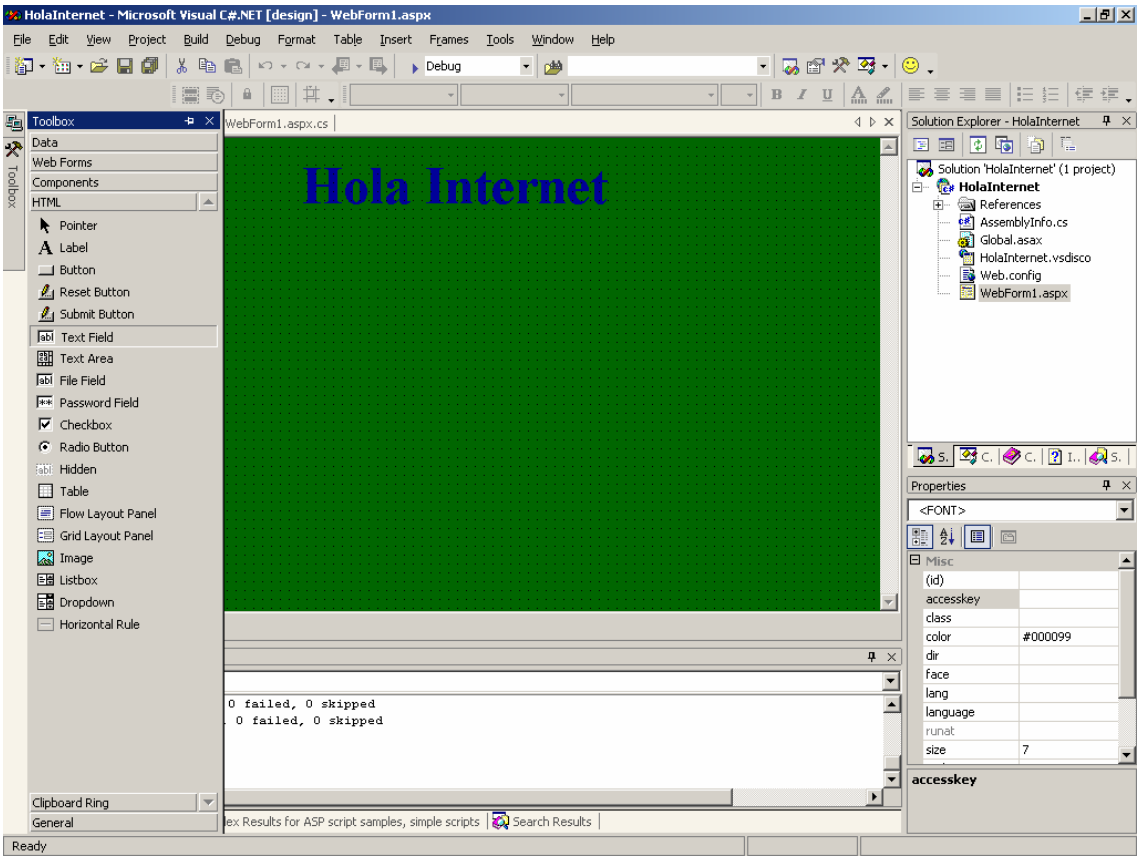

Figura 20.12. **Caja de herramientas**. Selección de un control HTML de tipo `Text Field`.

El resultado será:

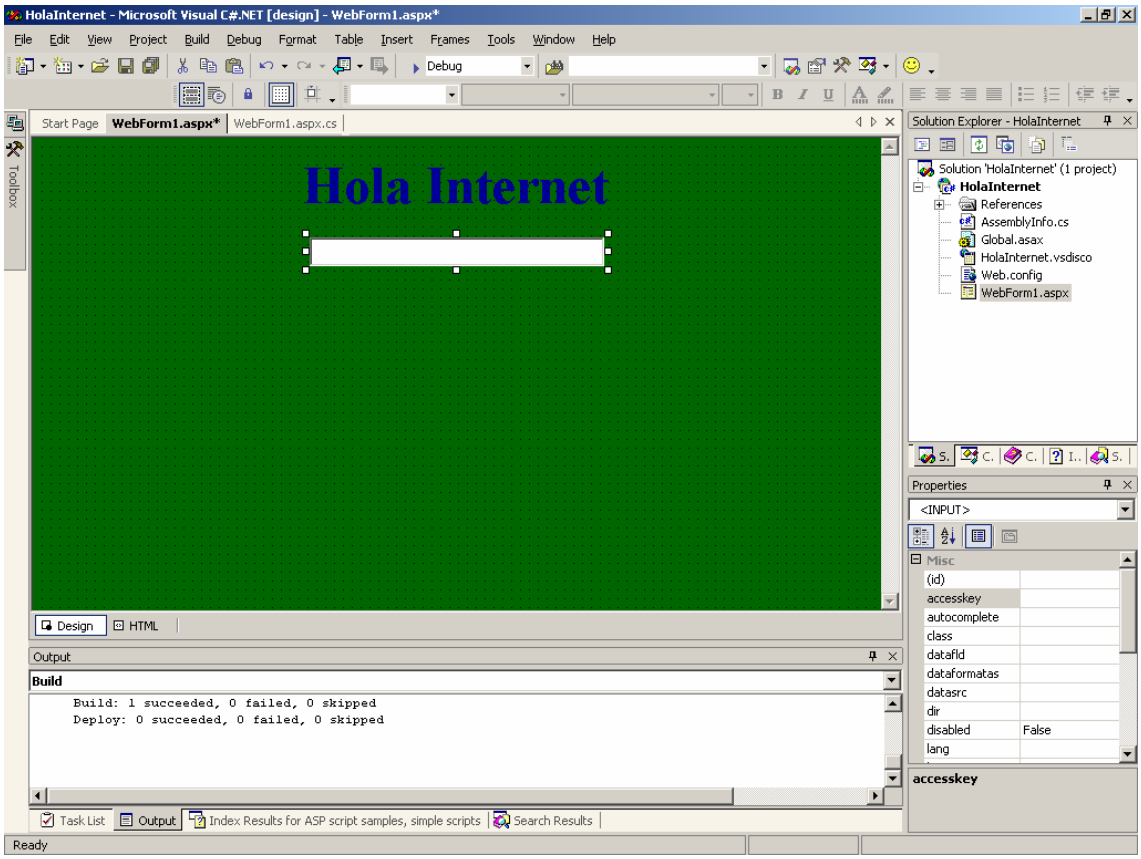

Figura 20.13. WebForm con la caja de Texto HTML. Vista diseño.

Donde el código HTML quedará como sigue:

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false" Inherits="HolaInternet.WebForm1" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<HTML>

      <HEAD>
            <meta name="GENERATOR" Content="Microsoft Visual Studio
                                                      7.0">
            <meta name="CODE_LANGUAGE" Content="C#">
            <meta   name="vs_defaultClientScript"   content="JavaScript
            (ECMAScript)">
            <meta                                  name="vs_targetSchema"
            content="http://schemas.microsoft.com/intellisense/ie5">
      </HEAD>

<body MS_POSITIONING="GridLayout" bgColor="#006600">

      <form id="Form1" method="post" runat="server">
            <h1 align="center">
                  <font color="#000099" size="7">Hola Internet</font>
            </h1>
            <INPUT style="Z-INDEX: 101; LEFT: 250px; WIDTH: 266px;
            POSITION: absolute; TOP: 90px; HEIGHT: 26px" type="text"
            size="39">
      </form>
```

```
</body>

</HTML>
```

Una vez se ha añadido el control se ha de mostrar su menú contextual y elegir **Ejecutar como control del servidor** para que sea un control HTML de Servidor:

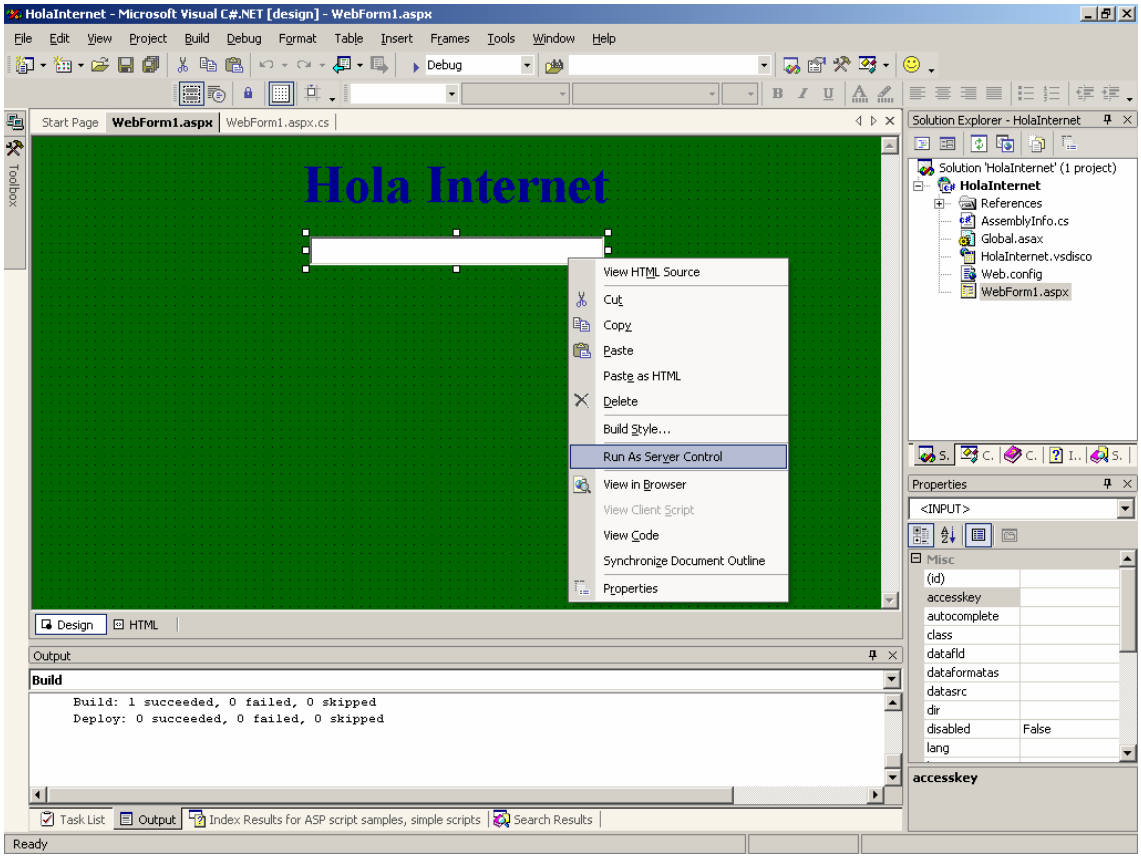

Figura 20.14. Conversión del control caja de texto HTML a control de servidor.

El código HTML habrá cambiado (se indicará que la caja de texto es de servidor y se le asignará in ID y un nombre):

```
<INPUT style="Z-INDEX: 101; LEFT: 250px; WIDTH: 266px; POSITION:
absolute; TOP: 90px; HEIGHT: 26px" type="text" size="39" id="Text1"
name="Text1" runat="server">
```

Para añadir un botón ASP.NET se ha de seleccionar la opción WebForms del **cuadro de herramientas**:

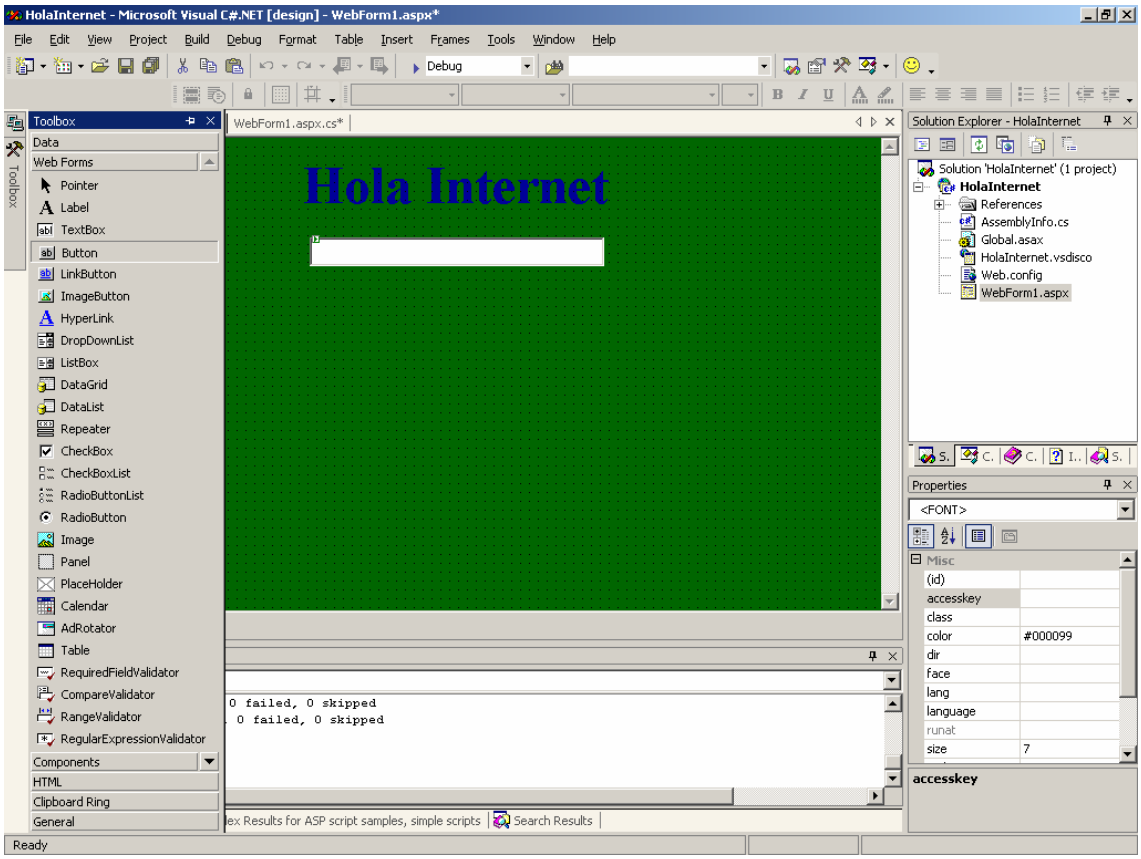

Figura 20.15. **Cuadro de herramientas**. Selección de un control ASP.NET (WebForms) de tipo `Button`.

El resultado será:

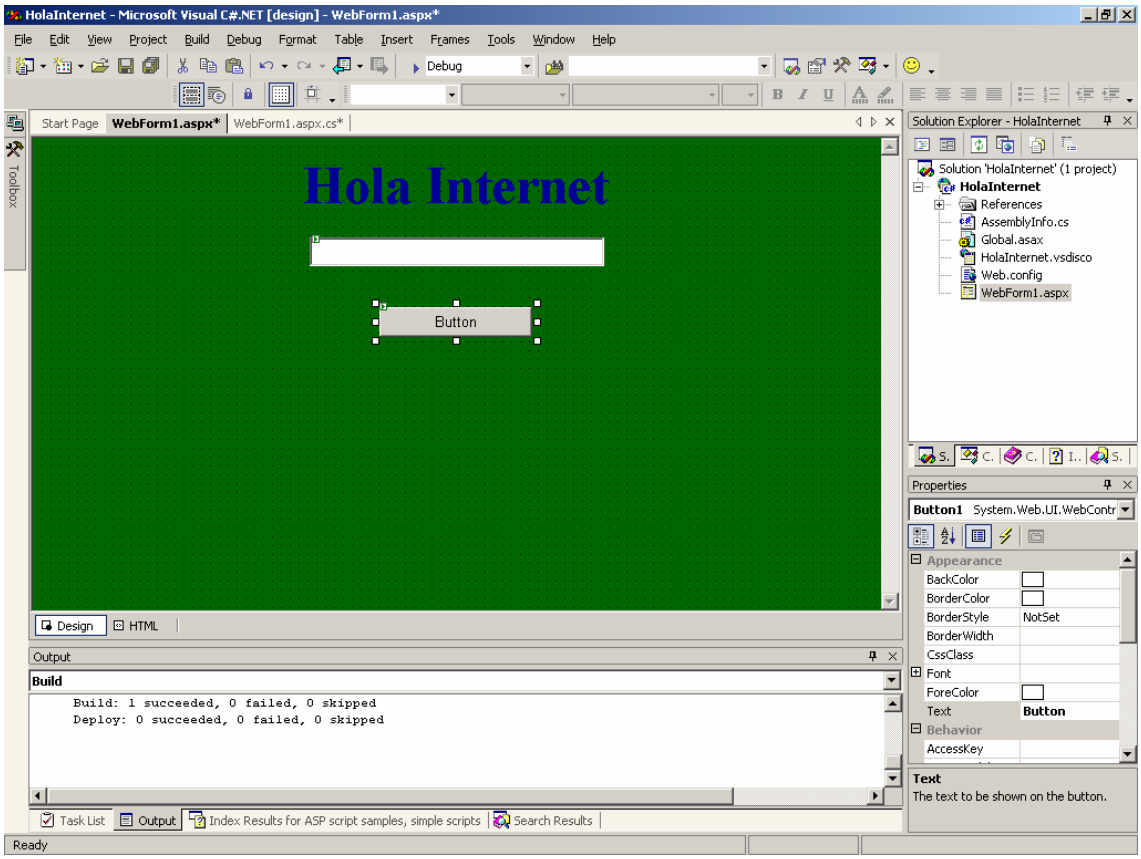

Figura 20.16. WebForm con el control `Button` ASP.NET. Vista diseño.

Y el código HTML:

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false" Inherits="HolaInternet.WebForm1" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<HTML>

      <HEAD>
            <meta name="GENERATOR" Content="Microsoft Visual Studio
7.0">
            <meta name="CODE_LANGUAGE" Content="C#">
            <meta name="vs_defaultClientScript" content="JavaScript
            (ECMAScript)">
            <meta name="vs_targetSchema"
            content="http://schemas.microsoft.com/intellisense/ie5">
      </HEAD>

<body MS_POSITIONING="GridLayout" bgColor="#006600">

      <form id="Form1" method="post" runat="server">

            <h1 align="center">
            <font color="#000099" size="7">Hola Internet</font>
            </h1>
```

```
            <INPUT style="Z-INDEX: 101; LEFT: 250px; WIDTH: 266px;
            POSITION: absolute; TOP: 90px; HEIGHT: 26px" type="text"
            size="39" id="Text1" name="Text1" runat="server">

            <asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 313px;
            POSITION: absolute; TOP: 153px" runat="server"
            Text="Button" Width="138px" Height="27px"></asp:Button>

      </form>

</body>

</HTML>
```

Es importante notar que la etiqueta del botón es `asp:Button`, la cual no corresponde a etiqueta HTML alguna. Cuando se ejecute la página, el elemento Botón se instanciará como un control ASP.NET de tipo `Button`.

Creación de manejadores de eventos.

Los controles de los WebForms pueden lanzar eventos. Por ejemplo, el control `Button` ASP.NET puede lanzar el evento `Click`. Este evento se dará en el cliente pero ha de ser manejado en el servidor, ya que el botón es un control de servidor.
Cuando el usuario pulsa el botón, la página es enviada al servidor y éste examina la información del evento. Si existe un método manejador de eventos para el evento que se ha dado, se llama a su código. Cuando el código del manejador del evento acaba, se envía la página de vuelta al cliente con los cambios realizados por el manejador del evento.

Ejemplo: A continuación se va a implementar una aplicación que muestre `Hola Internet` en la caja de texto cuando se pulse el botón.

El primer paso es hacer doble click sobre el botón (en modo diseño) para crear el manejador.

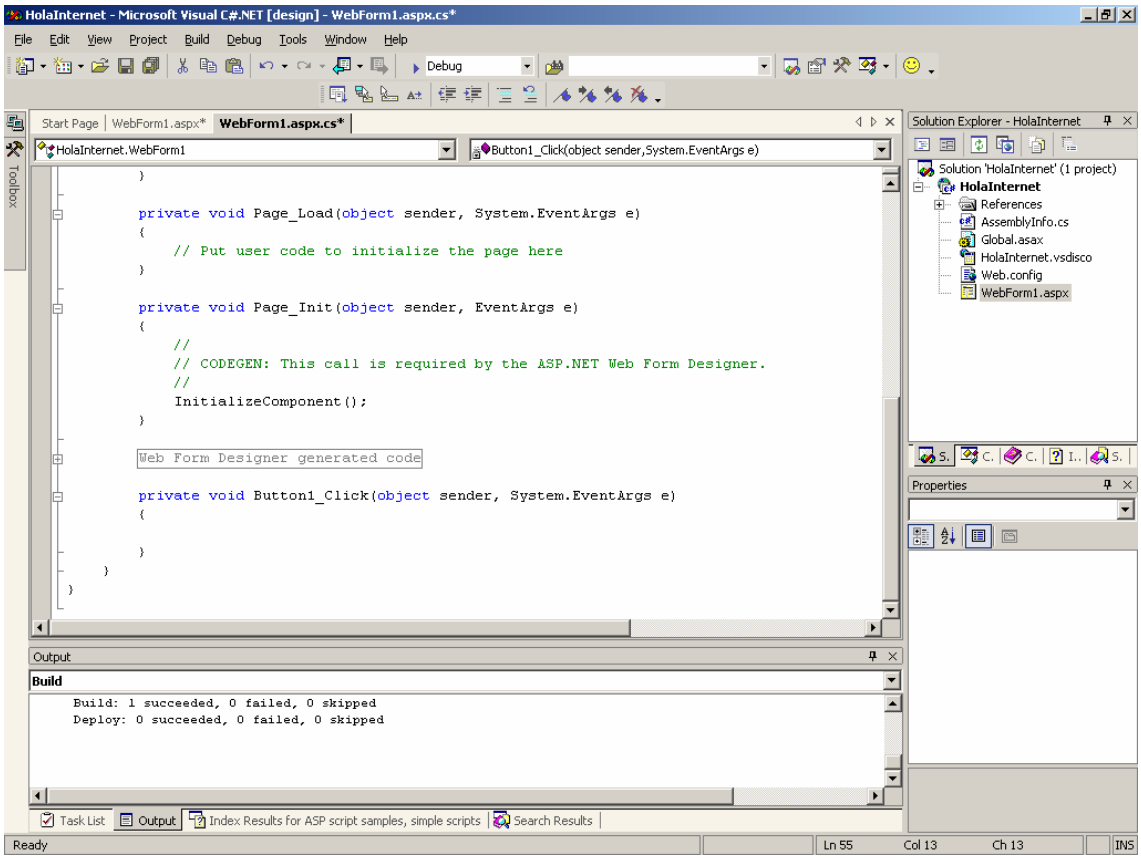

Figura 20.17. Código asociado al WebForm. Método de respuesta al evento (manejador) `Click` sobre el botón.

Es importante notar que se ha saltado al fichero `WebForm1.aspx.cs` que es el que contiene el código asociado a la página `WebForm1.aspx`. El código del fichero `WebForm1.aspx.cs` es:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace HolaInternet
{
	/// <summary>
	/// Summary description for WebForm1.
	/// </summary>
	public class WebForm1 : System.Web.UI.Page
	{
		protected System.Web.UI.HtmlControls.HtmlInputText Text1;
		protected System.Web.UI.WebControls.Button Button1;

		public WebForm1()
		{
```

```
            Page.Init += new System.EventHandler(Page_Init);
        }

        private void Page_Load(object sender, System.EventArgs e)
        {
            // Put user code to initialize the page here
        }

        private void Page_Init(object sender, EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web
            //Form Designer.
            //
            InitializeComponent();
        }

        #region Web Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.Button1.Click              +=              new
            System.EventHandler(this.Button1_Click);
            this.Load += new System.EventHandler(this.Page_Load);
        }
        #endregion

        private void Button1_Click(object sender, System.EventArgs
        e)
        {

        }
    }
}
```

Su puede observar que se han añadido las referencias:

```
protected System.Web.UI.HtmlControls.HtmlInputText Text1;
protected System.Web.UI.WebControls.Button Button1;
```

El siguiente paso es añadir el código del manejador:

```
public void Button1_Click (object sender, System.EventArgs e)
{
            Text1.Value = "Hola Internet";
}
```

Una vez hecho esto, sólo falta probarlo (con o sin debug):

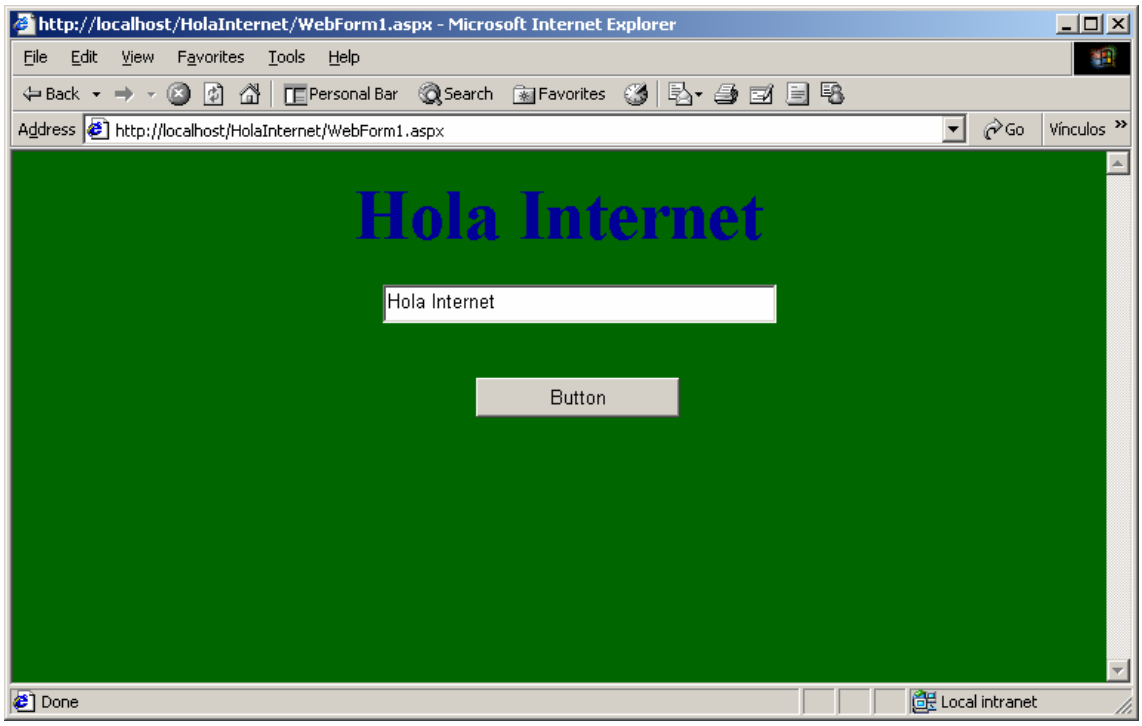

Figura 20.18. Ejecución de la aplicación. Vista tras pulsar el botón y lanzar el evento.