Argentina $8,90 (recargo al interior $0,20)) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net 9

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

ASP.NET Avanzado

Configuración - Autenticación
Páginas maestras

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Administrar el estado

Global.asax - ViewState
Sesión - Cookies

ISBN 978-987-1347-43-8
9 789871 347438 00009

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Controles de usuario Técnicas avanzadas

El uso de controles de usuario tiene una serie de dificultades que debemos sortear para lograr su correcto funcionamiento.

Persistencia y ViewState

El ViewState es una herramienta autogenerada por .NET, capaz de persistir los valores seleccionados en cada control de nuestro formulario Web, de manera automática.
En pocas palabras, todos los controles de servidor (incluyendo los de usuario) son persistidos en el ViewState, de modo tal que, cuando estamos del "lado del servidor", podemos acceder y modificar todas las propiedades publicas que los componen.
Esta capacidad, que parece funcionar automáticamente, también es adaptable a las necesidades de nuestra aplicación y podemos usarla en cualquier momento.

El ViewState resguarda la seguridad de los datos al encriptarlos y desencriptarlos de manera automática.

Persistiendo propiedades

El siguiente código representa una propiedad pública, persistida en el ViewState de nuestro User Control creado anteriormente (ucEjemplo). La propiedad MiPropiedad, de tipo String, será almacenada en el ViewState en el método Set, y recuperada en el método Get.

```
'CODIGO VISUAL BASIC 2005
Public Property MiPropiedad As String
   Get
      Return CStr(ViewState("MiPropiedad "))
   End Get
   Set
      ViewState("MiPropiedad ") = Value
   End Set
End Property
```

```
//CODIGO EN C#
public MiPropiedad Text {
   get {
      return (String) ViewState["MiPropiedad "];
   }
   set {
      ViewState["MiPropiedad "] = value;
   }
}
```

¿Código extraño en la página?

Si alguna vez visualizamos el código fuente de una página renderizada, es decir, el HTML resultante de la ejecución, habremos notado la inmensa cantidad de código añadido al original presente en Visual Studio. Parte de ese código generado (en general, la más llamativa e ilegible) es la representación encriptada del ViewState, conformada por un número de pares clave-valor con todas las propiedades de cada control en el formulario.

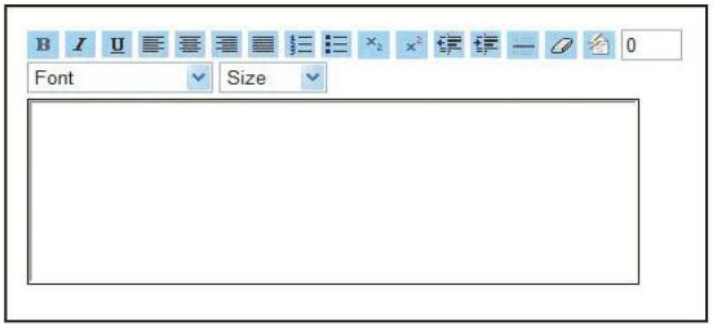


FIGURA 036 | RichTextBox, un control de usuario gratuito que permite editar texto HTML. Está disponible en www.codeproject.com.

De Web Forms a controles de usuario

Convertir todo un formulario Web en un control es una técnica sistemática, que puede sernos útil si necesitamos incluir toda la funcionalidad ya desarrollada en una página dentro de otra, o en un contenedor que tenga, además, otros controles.

Antes de nada, debemos renombrar el formulario a una extensión de control de usuario, es decir, reemplazar el .aspx por .ascx. A continuación, eliminamos del código HTML todas las etiquetas "inadecuadas" para un control de usuario; es decir, HTML, BODY y FORM. Acto seguido, reemplazamos la directiva @Page del encabezado de la página por la directiva @Control. A su vez, quitamos los atributos AutoEventWireup, CodeFile e Inherits, y sus correspondientes valores. Para finalizar, agregamos la propiedad className en la directiva del encabezado y establecemos un valor para el nombre de la clase. El código original de nuestra página era el siguiente:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Convertir.aspx.cs" Inherits=
"Convertir" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Página sin título</title>
</head>
<body>

  <form id="form1" runat="server">
  <div>
    <asp:TextBox ID="txtUsr" runat=
    "server"></asp:TextBox>
    <br />
    <asp:TextBox ID="txtPwd" runat=
    "server"></asp:TextBox><br />
    <asp:Button ID="btnLogin" runat="server"
    Text="Login" /></div>
  </form>

</body>
</html>
```

Transformado para ser un control de usuario, el código quedará de la siguiente manera:

```
<%@ Control Language="C#" ClassName=
"PruebaConversion" %>
  <div>
    <asp:TextBox ID="txtUsr" runat=
    "server"></asp:TextBox>
    <br />
    <asp:TextBox ID="txtPwd" runat=
    "server"></asp:TextBox><br />
    <asp:Button ID="btnLogin" runat="server"
    Text="Login" />
  </div>
```

¿Y el code behind?

En caso de que nuestro formulario Web posea code behind, en un archivo separado, tenemos que renombrar el archivo de código de ASPX.VB a ASCX.VB, y cambiar la forma en que hereda la clase que representa a la página; es decir que en vez de heredar de la clase Page, lo hará de UserControl.

Para finalizar, en los atributos del encabezado añadimos una propiedad llamada CodeFile y establecemos en el valor el nombre del archivo renombrado a ASCX.VB.

ASP.NET Avanzado

Programación Web para expertos

5

Contenidos

En el siguiente capítulo desarrollaremos algunas técnicas avanzadas vinculadas a la programación de sitios Webs utilizando ASP.NET.

Temas tratados

- Configuración
- Sistemas de autenticación
- Manteniendo el estado
- Páginas maestras
- Temas y skins
- Navegación por el sitio
- Acceso a datos

ASP.NET Avanzado

Técnicas, consejos y herramientas para desarrollar sitios Webs utilizando todo el potencial de este entorno.

Autenticación

En este apartado, veremos cómo autenticar a las personas para saber si las autorizamos o no a acceder a determinada información o a realizar alguna operación.

- Tipos de autenticación
- Autenticación por formularios
- Roles

Manteniendo el estado

Las aplicaciones ASP.NET necesitan del protocolo HTTP para poder comunicar el lado del cliente con el lado del servidor. Aprenderemos a trabajar con él.

- Almacenar el estado
- Los diferentes tipos de estado
- Global.asax
- Trabajar con cookies

Páginas maestras

Las páginas maestras son plantillas donde podemos definir el contenido común a todas las páginas, de manera sencilla y centralizada. Veremos cómo utilizarlas.

- Crear páginas maestras
- Aplicar la plantilla
- Ejemplos prácticos

Temas y skins

Veremos cómo aplicar todo el potencial de ASP.NET para lograr aplicaciones Web con una buena experiencia para el usuario.

- Crear y aplicar temas
- Cómo trabajar con skins
- Skins con ID

Acceso a datos

Conoceremos las posibilidades que brinda ASP.NET para el manejo de datos en nuestros formularios Web.

- Data binding
- Controles de origen de datos
- Parametrizando las consultas
- Controles para edición

Configuración de ASP

Iniciaremos el capítulo con algunas cuestiones especiales que hacen a la configuración del framework .NET.

El framework .NET provee soporte para gestionar la configuración tanto de la aplicación (parámetros necesarios para configurar nuestro sistema) como del motor de ejecución en sí. Podemos realizar toda la configuración a través de archivos en formato XML, lo que nos facilita aún más la tarea de modificar parámetros, ya que sólo precisaremos un simple editor de texto. Desde la versión 2.0, ASP.NET nos permite editar parámetros de configuración de la aplicación desde la ventana de propiedades en Internet Information Server (IIS), y esto ayuda mucho más a las tareas de administración.

Si bien el formato de los archivos de configuración es XML, no es necesario tratarlos como documentos de ese tipo, ya que el framework provee clases especiales para acceder a los valores de configuración. Estas clases serán repasadas más adelante.

Con respecto a ASP.NET, hay dos archivos de configuración principales que influyen en el modo en que se ejecutará la aplicación. Uno, llamado web.config, contiene la configuración propia de la aplicación; el otro archivo, machine.config (C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG\), incluye parámetros globales para el motor de ejecución de ASP.NET. Al ser globales, estos parámetros afectarán a cualquiera de las aplicaciones Web que corran en esa máquina. Debemos tener en cuenta que si bien el archivo machine.config es de este tipo, cualquiera de los parámetros que en él están definidos pueden ser sobrescritos en el archivo web.config, y al momento de la ejecución, se tomarán de ahí y no de la configuración global (es decir, el archivo web.config tiene prioridad sobre machine.config).

> En los archivos de configuración puede haber información sensible, de manera que deben estar bien resguardados de los usuarios maliciosos.

Para ejecutar una aplicación ASP.NET, no es necesario que el archivo web.config exista, porque muchos de los parámetros están en machine.config. Sin embargo, si estamos trabajando con Visual Studio 2005, éste nos exigirá tener un web.config para depurar la aplicación, y nos dará la opción de agregarlo automáticamente al momento de ejecutar la aplicación Web.

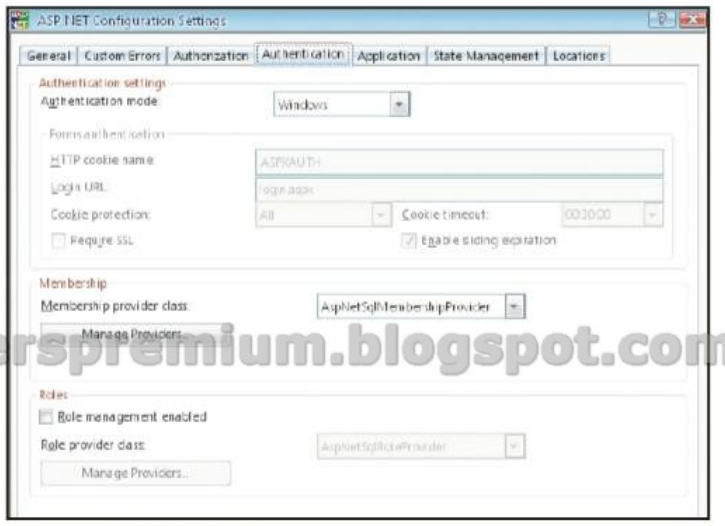


FIGURA 001 | Desde IIS es posible editar los parámetros de las aplicaciones ASP.NET.

Todo archivo .config es un documento XML, y como tal, se deben respetar los estándares XML, como tener un único elemento raíz, apertura y cierre de tags.

El archivo web.config

Como vimos hasta al momento, el archivo web.config nos permite configurar parámetros propios de la aplicación, así como sobrescribir valores de la configuración del motor de ejecución. En lo que respecta al comportamiento del motor de ejecución para nuestra aplicación, mediante este archivo, podemos configurar parámetros tales como el tipo de autenticación por utilizar, la duración de la sesión, qué usuarios tienen permiso de acceso a determinados recursos, y decenas de valores más. Si bien con un archivo de configuración por aplicación es suficiente, es muy común utilizar más de uno, distribuidos en las diferentes subcarpetas de la estructura de archivos. De este modo, es posible tener configuraciones personalizadas para las diferentes áreas de la aplicación. Uno de los usos más habituales de esta técnica es especificar distintos parámetros de seguridad y de acceso en cada subcarpeta o grupo de formularios Web.

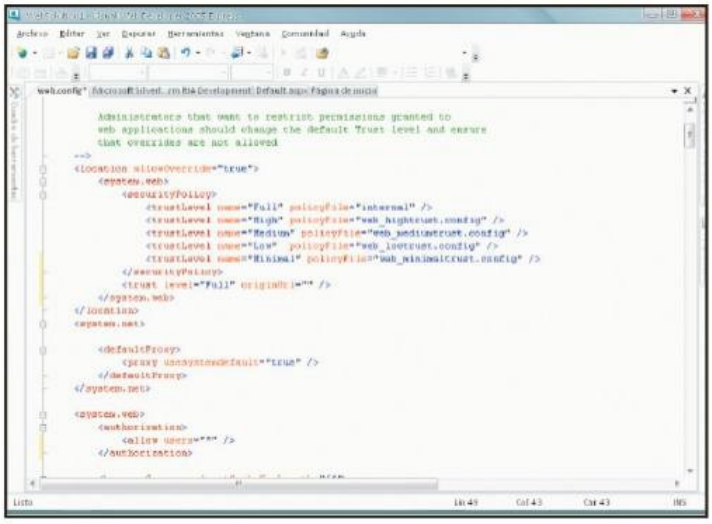


FIGURA 002 | Visual Studio provee intellisense también para el archivo web.config.

El archivo web.config consta de tres secciones bien definidas, todas dentro de un elemento raíz llamado **configuration**. En la Tabla 1 está definido cada uno de ellos.

Creación de parámetros propios

Los parámetros propios de nuestra aplicación deben colocarse dentro de la sección **appSettings**, que consta de una colección de elementos de tipo nombre-valor (key-value, en inglés). Esto significa que cada parámetro que agreguemos tendrá un nombre o clave y, por supuesto, su valor. El objetivo del nombre es tener una única forma de referenciarlo luego en el código de la aplicación para leer su valor.

Tabla 1 | Secciones del archivo web.config

Archivo: web.config		
Elemento: configuration		
appSettings	**connectionStrings**	**system.web**
Aquí colocamos los parámetros que necesitamos utilizar en la aplicación.	Aquí colocamos las cadenas de conexión a bases de datos, opción de encriptación, etc.	Configuración de los parámetros propios del motor de ejecución ASP.NET.

Para ilustrar el uso de los parámetros propios, imaginemos que nuestra aplicación permite subir archivos que se almacenan en el servidor para, luego, ser procesados. Si hacemos nuestra tarea bien, la ubicación en el disco donde se guardarán esos archivos tiene que ser configurable. Para lograrlo, agregamos un parámetro en la sección **appSettings** del archivo web.config:

```
<appSettings>
  <add key="RutaArchivosSubidos"
  value="C:\Documentos\Subidos"/>
</appSettings>
```

En el ejemplo anterior, hemos creado un parámetro llamado RutaArchivosSubidos, al que le asignamos el valor C:\Documentos\Subidos. De esta misma forma, podemos colocar todos los parámetros necesarios.

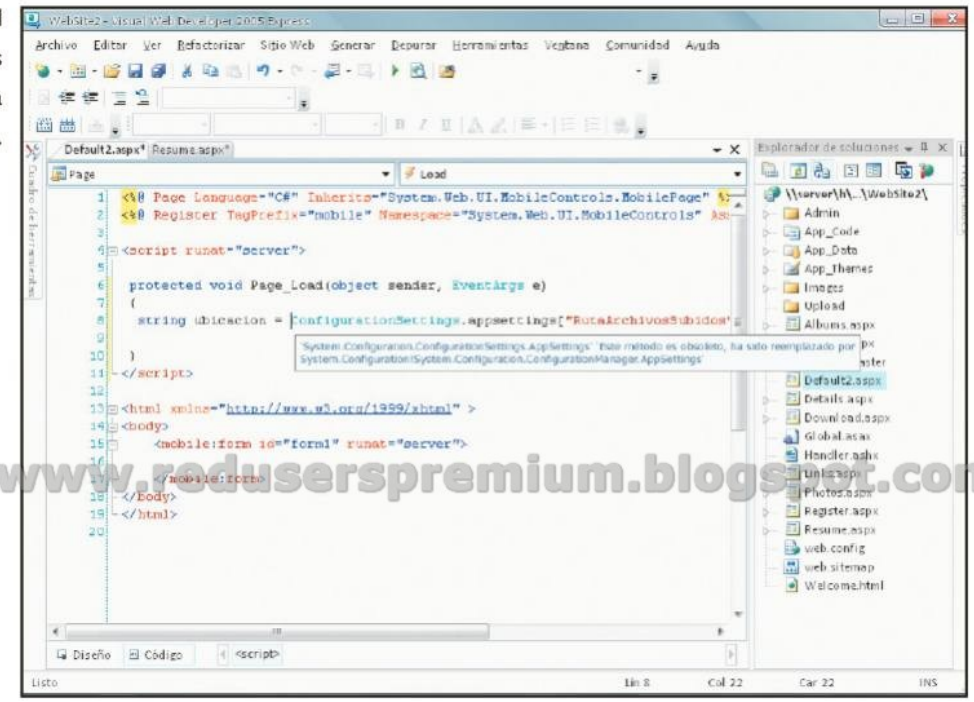
Acceso a la configuración

Ahora bien, de nada sirve crear parámetros en el archivo de configuración si luego no podremos leerlos. El framework .NET ofrece todo lo necesario para leer los parámetros de la configuración de manera simple y segura, abstrayendo detalles específicos como la ubicación del archivo o los tags XML.

> .NET ofrece todo lo necesario para leer los parámetros de la configuración de manera segura.

En las primeras versiones de ASP.NET, podíamos utilizar una clase llamada **Configuration Settings**, dentro del espacio de nombres **System.Configuration**, para acceder a la sección **AppSettings**. En la versión 2.0 esta clase fue marcada como obsoleta, y se incluyó una nueva, llamada **ConfigurationManager**, que brinda un mayor y mejor soporte para acceder a los distintos archivos y elementos de configuración. Por ejemplo, con ella podemos acceder a la configuración a nivel de máquina (el archivo machine.config).

FIGURA 003 | La propiedad ConfigurationSettings.AppSettings fue marcada como obsoleta en ASP.NET 2.0.



Volviendo al ejemplo de la ruta de almacenamiento de los archivos, para poder obtener el valor configurado, utilizamos la propiedad AppSettings de la clase ConfigurationManager. Ésta es una colección de valores a los que se puede acceder tanto por el nombre (la key que asignamos en el archivo web.config) como por su posición. Una buena práctica es acceder siempre por el nombre o key del parámetro, debido a razones de compatibilidad y claridad del código. Es más claro un nombre que un número; además, no nos limita a agregar nuevos valores insertándolos entre los que ya existen.
Entonces, para leer el valor del parámetro RutaArchivosSubidos, escribimos lo siguiente:

```
// C#
string ruta = ConfigurationManager.
AppSettings["RutaArchivosSubidos"];
```

```
// VB.Net
Dim ruta As String = ConfigurationManager.
AppSettings("RutaArchivosSubidos");
```

Esto es todo lo que necesitamos para acceder a los valores configurados dentro de la sección appSettings del archivo web.config. Es importante tener en cuenta que el tipo de dato de los valores de AppSettings es string. En este caso, no precisamos hacer nada porque estamos trabajando con una cadena de texto, pero para valores de otros tipos de datos, tendremos que hacer la conversión correspondiente.

Configuraciones avanzadas

Si bien la sección **appSettings** nos permite configurar muchos de los parámetros necesarios en nuestra aplicación, en ocasiones, el esquema clave-valor resulta muy simple e insuficiente para cubrir requerimientos más avanzados y complejos. Imaginemos que queremos configurar una ruta diferente para cada tipo de archivo que se pueda subir al servidor (para simplificar, asumamos que sólo se permiten archivos .doc, .pdf y .xls) y que, además, queremos poder determinar, por cada tipo de archivo, si se permite sobrescribir otros existentes.

FIGURA 004 | Durante la ejecución, podemos leer los parámetros y utilizarlos en nuestra aplicación.

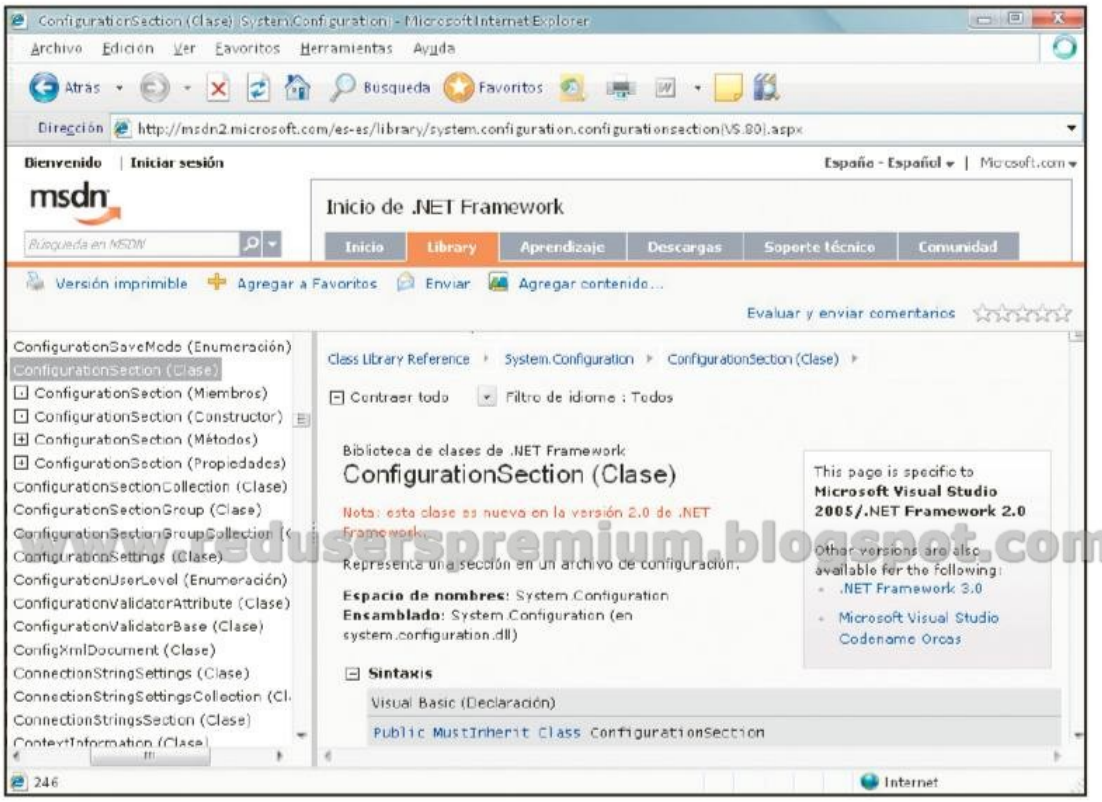
HAY TENER EN CUENTA QUE EL TIPO DE DATO DE LOS VALORES DE APPSETTINGS ES STRING. PARA LEER Y ALMACENAR VALORES DE OTROS TIPOS DE DATOS, DEBEREMOS HACER LA CONVERSIÓN CORRESPONDIENTE.

Obviamente, con pares clave-valor podríamos hacerlo, pero el resultado sería un archivo difícil de entender y de mantener, al igual que el código necesario para leer la configuración. Para solucionar problemas como éste, ASP.NET incluye un par de clases llamadas ConfigurationSection y ConfigurationElement, que nos permiten definir clases con las propiedades que queremos hacer configurables, especificando mediante atributos cómo se debe leer del archivo de configuración. Luego, simplemente usando el método GetSection de la clase ConfigurationManager, podemos leer nuestra configuración, con la ventaja de que ya tendremos un objeto con sus propiedades fuertemente tipadas. Los detalles acerca de cómo utilizar estas clases escapa al alcance de este curso, pero en la documentación de Visual Studio y en el sitio de MSDN es posible encontrar explicaciones y ejemplos.

La sección System.Web

Como vimos al principio, la sección System.Web permite configurar algunos parámetros del motor de ejecución de ASP.NET que afectan directamente el comportamiento de nuestra aplicación y su desarrollo. En próximos capítulos, veremos cómo utilizar los distintos elementos de esta sección para configurar temas tales como el tipo de autenticación o el tipo de almacenamiento de estado del lado del servidor. Además de parámetros para la ejecución, en la sección System.Web podemos configurar otros valores que hacen al desarrollo y la compilación de la aplicación, como los assemblies necesarios (recordemos que el nuevo modelo de web sites no posee un archivo de proyecto), el tipo base de todas las páginas, la página maestra por defecto, y otros valores más.

FIGURA 005 | En el sitio de MSDN hay más información sobre la clase ConfigurationSection.



Autenticación

Veremos cómo definir los permisos de usos de nuestro sistema a los diferentes usuarios.

Cuando desarrollamos aplicaciones medianas o grandes, vemos la necesidad de determinar quién es la persona que está usando el sistema, para saber si lo autorizamos o no a acceder a cierta información o para realizar alguna operación. En estos casos, tendremos que apelar a dos conceptos fundamentales –autenticación y autorización–, y ponerlos en práctica.

¿Qué entendemos por autenticación?

Autenticar es el acto de determinar que algo es auténtico o verídico, es decir, que realmente es lo que dice ser. Así como nosotros somos identificados o autenticados en el mundo real a través de nuestro documento para realizar cualquier trámite, en el campo de la seguridad informática podemos decir que **autenticación** es el proceso de determinar que la identidad de alguien (o de algo) es auténtica, que esa autenticidad puede ser verificada por las políticas de seguridad del sistema y que la identidad indicada se corresponde con una de las conocidas.
Si bien existen otras formas más complejas (y seguras) de autenticación, la más común es mediante el uso de un nombre de usuario y una contraseña o password. El usuario se identifica a través de su nombre y confirma su identidad mediante la contraseña. El sistema posee un

Proceso de autenticación

FIGURA 006 | En el proceso de autenticación, el sistema se asegura de que la identidad del usuario sea válida.

registro de los usuarios conocidos, cada uno con su respectiva contraseña. Cuando el usuario provee ambos valores, si éstos coinciden con los almacenados, el sistema puede concluir que la identidad es **auténtica**, es decir que el usuario ha sido autenticado. En las próximas secciones estudiaremos las diferentes herramientas que nos ofrece ASP.NET para determinar la identidad del usuario y cómo autenticarlo.

Una vez que hemos autenticado la identidad del usuario, estamos en condiciones de verificar si está autorizado para realizar la operación que intenta hacer o si tiene acceso a los datos que ha solicitado. Este proceso se conoce como **autorización**. Obviamente, no podemos autorizar a alguien que no conocemos; es por eso que la autenticación es lo primero que debemos hacer para llevar a cabo la autorización.

Autenticación en ASP.NET

En ASP.NET encontramos diferentes opciones en cuanto a tipos de autenticación: por formularios, integrada de Windows y Passport. Veamos brevemente cada una y, luego, estudiaremos la primera con más detalle.

Autenticación por formularios

En este tipo de autenticación, el desarrollador es responsable de solicitar al usuario sus credenciales para poder validar su autenticidad. Debemos crear una página en la aplicación donde el usuario pueda ingresar su nombre y contraseña y, luego, tenemos que utilizar esos valores para verificar que correspondan con los de un usuario conocido.

Autenticación integrada de Windows

Con este tipo de integración, el navegador se encarga de recolectar las credenciales de usuario y enviarlas al servidor. El programador no necesita hacer nada para validar el usuario. En general, se utiliza en entornos controlados (por ejemplo, en intranets), donde la validación se

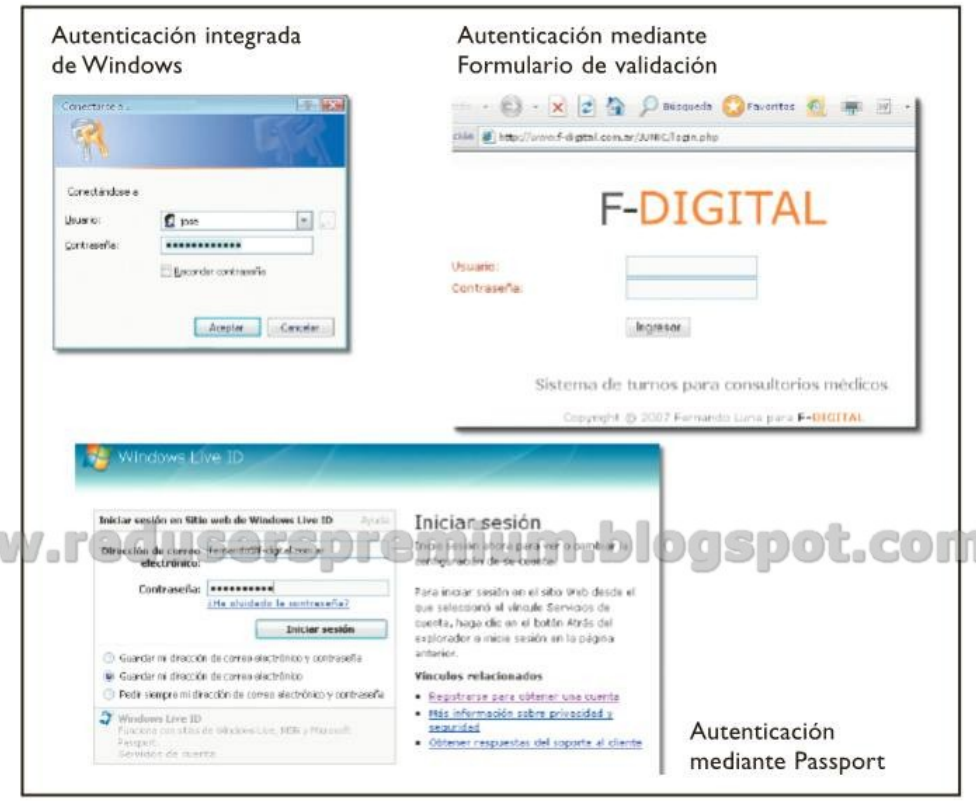


FIGURA 007 | Ejemplo de los tres tipos de autenticación mencionadas en la Tabla 2.

realiza contra algún servidor dedicado a tal efecto, como puede ser un Active Directory de Windows.

Autenticación por Passport

Microsoft ofrece un servicio global de autenticación que permite identificarse en múltiples sitios Web mediante una única cuenta o nombre de usuario. Además, el almacenamiento de los datos de los usuarios se efectúa en servidores provistos por la empresa, lo que permite que los desarrolladores se desliguen de la tarea de almacenar y recuperar los datos de los usuarios. Quizá por no ser un servicio gratuito, Passport no tuvo una gran aceptación como alternativa de autenticación.

Cada tipo de autenticación que nos ofrece ASP.NET tiene sus ventajas y desventajas. Veamos algunas de ellas en la Tabla 2.

Elegir el tipo de autenticación

Entrando en el desarrollo de una aplicación con soporte para autenticación en ASP.NET, una de las primeras tareas que debemos hacer es seleccionar el tipo de autenticación que vamos a utilizar, evaluando las ventajas y desventajas de cada uno y qué necesitamos en nuestra aplicación. Una vez que hayamos tomado una decisión, debemos indicárselo al motor de ejecución de ASP.NET mediante el archivo de configuración de nuestra aplicación (web.config). Para hacerlo, contamos con un elemento especial que podemos colocar dentro de system.web, denominado **authentication**. Éste posee un atributo llamado **mode**, en el que deberemos especificar el tipo de autenticación. Por ejemplo, podríamos tener el archivo de configuración que se muestra a continuación.

Tabla 2 | Tipos de autenticación

Mediante formulario	Integrada de Windows	Mediante Passport
Quizás es la que más flexibilidad y libertad nos da a la hora de decidir cómo y dónde pedir las credenciales al usuario, como así también para la creación y baja de éstos. Como desventaja podemos citar que deberemos ser nosotros quienes implementemos todo lo relacionado a la autenticación.	Ventaja: Es la más fácil de utilizar y la más segura. Es segura porque, al momento de enviarse las credenciales al servidor, no se manda el nombre de usuario y la contraseña, sino un "token" de seguridad creado por el usuario (cuando se logueó en Windows). Por lo tanto, si alguien analiza los paquetes que viajan por la red, no verá información sensible, como la contraseña. Desventaja: Dificulta la gestión de usuarios en aplicaciones públicas en Internet, ya que los usuarios deben ser dados de alta desde fuera de la aplicación.	Ofrece la ventaja de que el usuario no tiene que recordar diferentes nombres (y, posiblemente, contraseñas) para cada uno de los sitios que visita. Sólo deberá registrarse en el primero y, luego, mantendrá sus credenciales mientras no cierre el navegador. La gran desventaja de Passport es que, al ser un servicio de terceros, estamos atados a la voluntad del proveedor, además de al costo que tiene utilizarlo.

```
<?xml version="1.0"?>
<configuration>
  <appSettings />
  <connectionStrings/>
  <system.web>
      <authentication mode="Windows" />
  </system.web>
</configuration>
```

Aquí le estamos indicando a ASP.NET que queremos utilizar autenticación integrada de Windows. Los valores que puede tomar el atributo mode son **None** (sin autenticación), **Forms** (por formularios), **Windows** (integrada de Windows) y **Passport**.

Autenticación por formularios

Veremos a continuación cómo trabajar con autenticación por formularios. En las versiones 1.0 y 1.1 del framework .NET, toda la lógica de administración de usuarios y control de login debía ser implementada por nosotros mismos. Desde la versión 2.0, se incluyó un excelente soporte para el trabajo con autenticación de usuarios (aunque si lo deseamos, también podemos hacerlo en forma manual, como en las versiones anteriores). Estudiaremos primero los conceptos fundamentales que se aplican a cualquiera de las versiones y, luego, veremos cómo hacerlo en la versión 2.0.

Configurando la autenticación

Lo primero que debemos hacer es configurar la aplicación para que use autenticación por formularios. Esto se hace en la sección **authentication** del archivo web.config, utilizando el valor Forms. Si ejecutamos la aplicación así como está, veremos que no ocurre nada nuevo.

Para ver cómo funciona la seguridad en ASP.NET, vamos a prohibir el acceso de

FIGURA 008 | El sitio de DCE utiliza autenticación por Passport.

EL NOMBRE LOGIN.ASPX ES EL VALOR POR DEFECTO UTILIZADO POR ASP.NET COMO PÁGINA DE LOGIN O IDENTIFICACIÓN DE USUARIO. PARA CAMBIARLO, DEBEMOS ESPECIFICARLO COMO UN ATRIBUTO DEL ELEMENTO FORMS DENTRO DEL ELEMENTO AUTHENTICATION, EN EL ARCHIVO WEB.CONFIG.

usuarios anónimos a nuestra aplicación. En este caso, vamos a utilizar otra sección del archivo de configuración: **authorization**, que nos permite especificar el nivel de acceso a la aplicación. Por ejemplo, podemos dar acceso a todos los usuarios o no permitir usuarios anónimos.

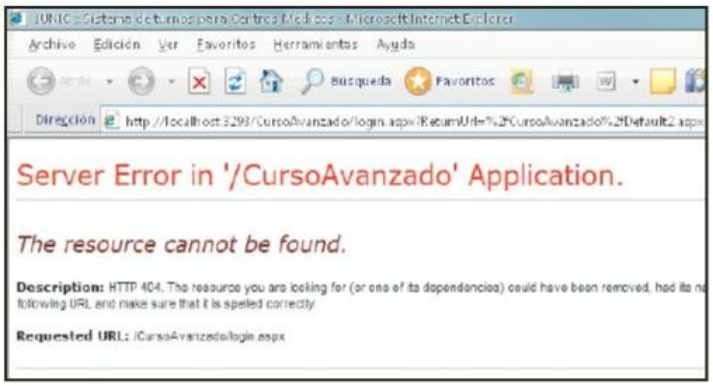


FIGURA 009 | Si no creamos la página de login, obtendremos un error cuando el sitio requiera que nos identifiquemos como usuario.

En nuestro ejemplo, vamos a negar el acceso a los usuarios anónimos:

```
<authorization>
  <deny users="?" />
</authorization>
```

El signo de interrogación es el símbolo que se utiliza para especificar usuarios anónimos. Mediante el elemento **deny** (denegar), estamos prohibiendo el acceso a los usuarios no autenticados. Tengamos en cuenta que dentro del elemento authorization podemos colocar varias cláusulas de seguridad.
En el ejemplo siguiente, estamos negando el acceso a los usuarios anónimos y al usuario Juan, pero lo permitimos a todos los que tengan el rol Gerentes (luego estudiaremos el concepto de rol).

Tabla 3 | Posibles combinaciones que se pueden utilizar

	Users		Roles	
	?	*	Nombre de usuario	Nombre de rol
allow	Permite el acceso a usuarios anónimos.	Permite el acceso a todos los usuarios.	Permite el acceso del usuario con el nombre indicado, que tenga el rol indicado.	Permite el acceso a cualquier usuario
deny	Prohíbe el acceso a usuarios anónimos.	Prohíbe el acceso de cualquier usuario.	Prohíbe el acceso del usuario con el nombre indicado.	Prohíbe el acceso de los usuarios que tengan el rol indicado.

```
<authorization>
  <deny users="?" />
  <deny users="Juan"/>
  <allow roles="Gerentes"/>
</authorization>
```

Si ejecutamos la aplicación, veremos que da un error diciendo que el recurso no se encuentra; se refiere al formulario **login.aspx**. El funcionamiento de la aplicación es el siguiente. Cuando intentamos acceder a una página, el motor de ASP.NET se fija, en la configuración, quiénes tienen acceso a ella. Luego, determina si estamos accediendo como un usuario anónimo o como uno autenticado, y cruza esta información con la configuración, para determinar si nos autoriza o no a acceder a la página. Si somos un usuario anónimo, y el acceso anónimo está denegado, el motor de ASP.NET direccionará automáticamente el browser a una página llamada login.aspx, para que ingresemos nuestras credenciales. Nosotros seremos los responsables de crear la página login.aspx con el contenido necesario para que el usuario indique sus credenciales y podamos autenticarlo.

Si ahora creamos un nuevo web form llamado **login.aspx**, le colocamos dos cajas de texto para que el usuario ingrese su nombre y contraseña, y ejecutamos la aplicación, veremos que, efectivamente, seremos redireccionados a este nuevo formulario.

Autenticación del usuario

Cuando el usuario ingresa sus datos en el formulario login.aspx, en el code behind debemos validar que éstos correspondan a los de un usuario; si es así, guardamos la información de login y redireccionamos a la página original (la que el usuario quiso ver cuando fue redirecionado al formulario de login). Afortunadamente, para realizar esta tarea, ASP.NET cuenta con un método que hace casi todo el trabajo por nosotros: Forms Authentication.RedirectFromLoginPage. Éste recibe como parámetro el nombre del usuario y crea una cookie en el browser para no perder la información de autenticación. Luego, utiliza el parámetro ReturnUrl (lo veremos en el cuadro de dirección de la Figura 011) para devolver al usuario a la página original. Para entender este concepto, agreguemos

FIGURA 010 | Mediante la configuración, podemos cambiar el nombre de la página de login que se va a utilizar.

```
<authentication mode="Forms">
  <forms loginUrl="Identificacion.aspx" | />
</authentication>

<authorization>
  <deny users="?" />
  <deny users="Juan"/>
  <allow roles="Gerentes"/>
</authorization>
<roleManager enabled="true" />
<compilation debug="true" defaultLang
<!--
        The <authentication> section enables configuration
        of the security authentication mode used by
        ASP.NET to identify an incoming user.
    -->
```

cookieless
defaultUrl
domain
enableCrossAppRedirects
lockAllAttributesExcept
lockAllElementsExcept
lockAttributes
lockElements
lockItem
name

ASP.NET NOS BRINDA UN MÉTODO PARA REDIRECCIONAR A LOS USUARIOS A LA PÁGINA DE LOGIN DEL SISTEMA, CUANDO ÉSTE ES REQUERIDO. SE LLAMA FORMSAUTHENTICATION.REDIRECTFROM LOGINPAGE Y HACE CASI TODO EL TRABAJO POR NOSOTROS.

las siguientes líneas en el archivo de code behind del formulario de login:

```
protected void btnAceptar_Click(object
sender, EventArgs e)
{
    FormsAuthentication.RedirectFromLogin
    Page(txtUsuario.Text, false);
}
```

Estas líneas capturan el evento clic del botón Aceptar y llevan a cabo el registro del usuario como identidad autenticada (asumamos que txtUsuario es el cuadro de texto donde el usuario ingresa su nombre). Al ejecutar la aplicación, veremos que, luego de ingresar un nombre cualquiera de usuario, nos redirecciona a la página a la que queríamos acceder en un principio.
Un detalle muy importante: si como nombre de usuario escribimos Juan, el browser se quedará en la página de login. ¿Por qué sucede esto? Bien, porque antes configuramos la aplicación para denegar el acceso al usuario Juan (deny users="Juan").
Vemos, entonces, que el motor de ASP.NET se encarga de la autorización de los usuarios de manera automática en función de lo que hayamos declarado en el archivo de configuración.

Obtener datos del usuario

Una vez que el usuario está autenticado, es muy probable que en el código de la aplicación necesitemos conocer su identidad, ya sea para mostrarla en la página, guardar un registro de auditoría o lo que sea. Para eso, el framework provee una interfaz llamada Principal; ésta, a su vez, posee una propiedad de tipo Identity, que nos brinda información sobre la identidad del usuario. Para acceder a estos datos, podemos hacerlo a través de la propiedad Context.User de la página:

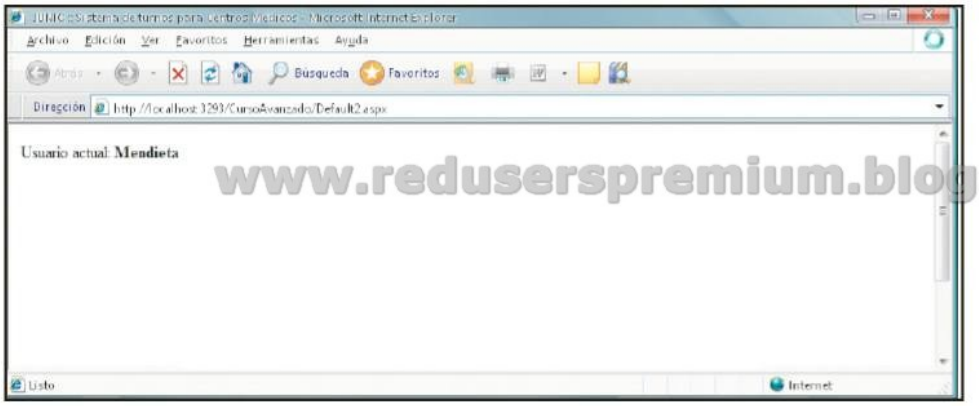


FIGURA 011 | Mediante Principal e Identity, podemos conocer quién es el usuario actual.

UN ROL PUEDE DEFINIRSE COMO UNA CARACTERÍSTICA QUE PUEDE TENER UN USUARIO O GRUPO DE USUARIOS, COMO GERENTES, ADMINISTRADORES, CONTADORES, ETC.

```
protected void Page_Load(object sender,
EventArgs e)
{
    if (Context.User.Identity.
    IsAuthenticated)
        Response.Write(Context.User.
        Identity.Name);
}
```

Como vemos en el código, mediante Identity podemos saber si el usuario está autenticado y cuál es su nombre. A través de Principal, sabremos también si está en un determinado rol.

Roles

El esquema de seguridad de .NET nos permite trabajar con el concepto de rol. Un **rol** puede definirse como una característica que puede tener un usuario o grupo de usuarios. Por ejemplo, podemos tener el rol Gerentes, Administradores o Contadores. El objetivo de los roles es facilitar la gestión de la autorización de usuarios, para no tener que habilitar o deshabilitar el acceso a cada uno de ellos. Así, podemos permitir o denegar accesos por rol, y reducir la cantidad de configuraciones y líneas de código necesarias para llevar a cabo la autorización (o no) a los distintos recursos de la aplicación. La interfaz IPrincipal posee un método llamado IsInRole, que podemos emplear para averiguar si el usuario está en un determinado rol.

Autenticación en ASP.NET 2.0

Como mencionamos antes, ASP.NET 2.0 incluye herramientas de autenticación mucho más elaboradas, que facilitan el trabajo del programador. Esto se logró mediante la inclusión de una batería de controles listos para usar. Entre ellos, hay uno para login, otro para mostrar el usuario actual e, incluso, uno para crear un nuevo usuario.

Junto a los nuevos controles, se introdujo el concepto de **provider de seguridad**. Se trata de clases que se encargan de toda la gestión de usuarios, como su almacenamiento y administración, validación y algunas tareas más. ASP.NET 2.0, por defecto, incluye un provider llamado AspNetProvider, que trabaja en conjunto con SQL Server. Veamos brevemente cómo utilizarlos.
Lo primero que debemos hacer es crear una cadena de conexión llamada LocalSqlSever, colocarla dentro del elemento **connectionStrings** del archivo de configuración y hacerla apuntar a un servidor de SQL Server. Luego, desde la consola de Visual Studio (a la que se accede yendo a Incio/Programas/Microsoft Visual Studio 2005/Visual Studio tools), ejecutamos el programa aspnet_regsql. Se abrirá un asistente que nos ayudará a crear las tablas y stored

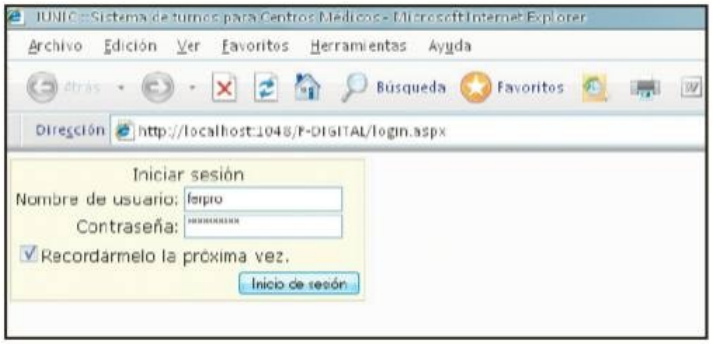


FIGURA 012 | El control de login nos ahorra mucho trabajo.

procedures que AspNetProvider necesita para gestionar los usuarios. Una vez hecho esto, vamos a Visual Studio y accedemos a la página de configuración de la aplicación, desde Microsoft Visual Web Developer Express 2005/Sitio Web/Configuración ASP.NET. Allí ejecutamos el asistente.

A través de este asistente, podemos elegir el tipo de autenticación y crear algunos usuarios y roles. Vamos a generar un usuario cualquiera para hacer la prueba. Volviendo a nuestro formulario login.aspx, borramos lo que habíamos creado (tanto en el diseñador como en el code behind) y arrastramos un control de login desde la barra de herramientas. Esto es todo lo que necesitamos hacer para que los usuarios puedan identificarse en el sistema. Todo lo demás está a cargo del control y el provider de seguridad. Si ejecutamos la aplicación, veremos que aparece el control de login, y al ingresar los datos del usuario que hemos creado anteriormente, pasaremos a la otra página, como un usuario autenticado; en caso contrario, el control de login mostrará un mensaje indicando que las credenciales no son válidas.

Una de las mayores ventajas de este esquema es que podemos implementar nuestro propio provider de seguridad para que trabaje según nuestras necesidades, configurarlo en el archivo web.config y utilizar los controles de login del mismo modo. Además, estos controles brindan propiedades y funcionalidad para la mayoría de las tareas relacionadas con la autenticación de usuarios (recuperación de clave, login automático para la próxima vez, creación de usuarios, etc.).

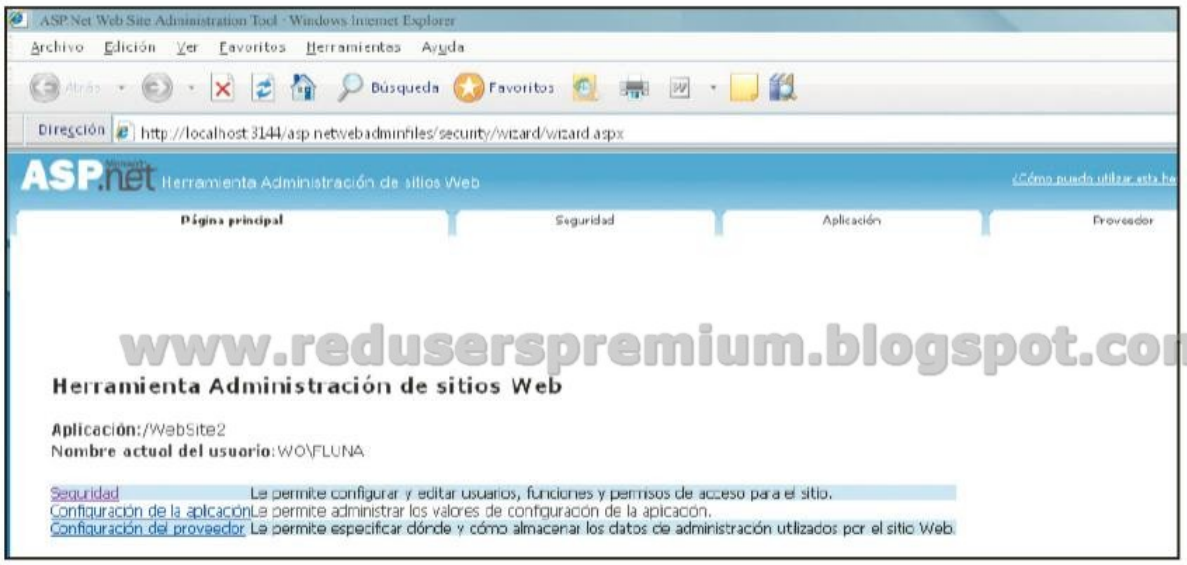


FIGURA 013 | Mediante la herramienta de configuración, podemos ajustar los parámetros de seguridad.

Mantener el estado

Veremos cómo trabajan las aplicaciones ASP.NET para comunicar el lado del cliente con el del servidor.

Las aplicaciones ASP.NET necesitan el protocolo HTTP para comunicar el lado del cliente con el del servidor. Este protocolo es, esencialmente, desconectado; esto significa que el cliente se conecta al servidor, pide el recurso que necesita (una página, una imagen, etc.), el servidor se lo entrega y se desconecta, con lo cual finaliza la comunicación.

En ASP.NET, cuando un usuario solicita una página, del lado del servidor se crea un entorno de ejecución y se asigna un espacio de memoria. Para atender el pedido del usuario, ASP.NET ejecuta el código generado a partir del diseño y del code behind de la página. Durante la ejecución, todos los objetos y las variables se colocan en el espacio de memoria que tiene asignada la solicitud o request.

Cuando el formulario termina de ejecutarse, se entrega el resultado al cliente y se libera la memoria utilizada, con lo cual se desechan todos los objetos creados. Por ejemplo, si en el code behind de la página declaramos una variable de tipo DataSet y la cargamos con datos que traemos de SQL Server, durante un postback posterior, la variable estará sin inicializar (en null o Nothing, dependiendo del lenguaje), porque, al terminar la ejecución anterior, fue desechada y, al comenzar la actual, fue declarada otra vez.

Con este esquema de ejecución, cuando la página se envía al cliente, perdemos cualquier valor que necesitemos almacenar para la próxima solicitud. Como en la mayoría de las aplicaciones se precisa mantener el estado de ciertos

Comunicación con el servidor

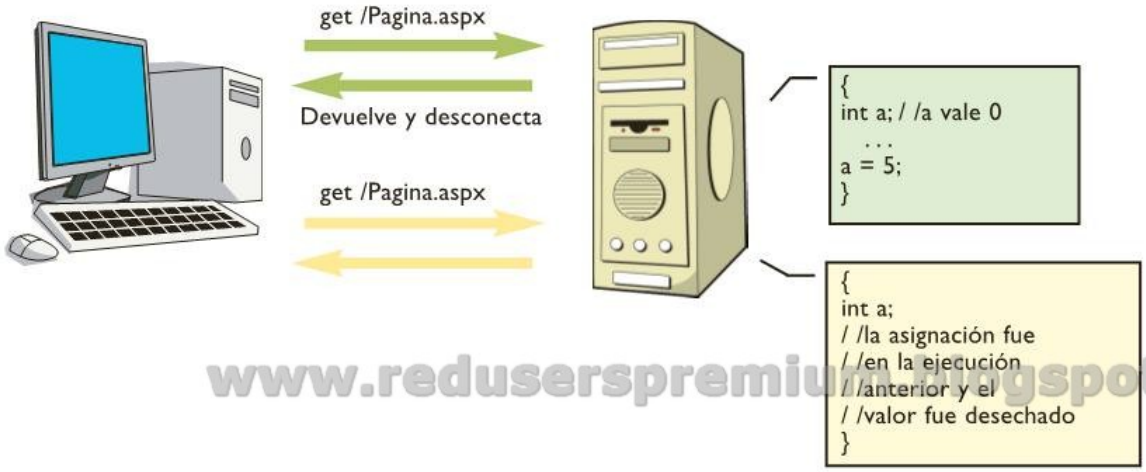


FIGURA 014 | Cuando la ejecución termina, la conexión se corta y las variables se desechan. Al siguiente pedido, las variables tendrán otra vez su valor inicial.

Para utilizar el estado de la aplicación desde fuera de una clase que herede desde Page, deberemos hacerlo a través de HttpContext.Current.Application.

valores de la ejecución actual, ASP.NET incluye algunas herramientas para facilitarnos la tarea. Así es que podremos almacenar datos en la memoria del servidor o en el cliente, dependiendo de los requerimientos de cada caso. A continuación, estudiaremos cuáles son las alternativas con las que contamos, y cómo utilizar cada una y sacar el mayor provecho.

Almacenar estado del lado del servidor

La memoria del servidor permite mantener el estado de la aplicación. Existen tres tipos de almacenamiento en este caso: el estado de la aplicación, el estado de la sesión y la caché; veamos cada uno.

¿Del lado del cliente o del servidor?

ASP.NET nos permite mantener el estado de la ejecución tanto del lado del cliente como del lado del servidor. Pero a la hora de almacenar un valor para no perder el estado, deberemos evaluar cuidadosamente cómo y dónde guardarlo, porque cada una de las alternativas tiene sus ventajas y desventajas, y afecta el rendimiento y la escalabilidad de la aplicación.

El estado de la aplicación

Es un espacio de memoria compartido por todos los usuarios de la aplicación; hay uno por aplicación ASP.NET. Aquí podremos guardar cualquier valor y, luego, recuperarlo en algún postback o en una página distinta. Al ser un espacio compartido, debemos tener cuidado con la manipulación, ya que lo que se ejecuta para un usuario no debe afectar negativamente la ejecución de los demás.

Para acceder al estado de la aplicación, la clase Page expone una propiedad llamada Application, de tipo HttpApplicationState (recordemos que todos los web forms heredan de Page). Este tipo representa una colección diccionario o nombre-valor; es decir que para agregar elementos, debemos asociarles un nombre. Luego, para recuperar el valor, lo hacemos a través del nombre. Los valores que podemos almacenar en el estado de la aplicación son de tipo **Object**; o sea que podremos guardar cualquier cosa, pero teniendo en cuenta que, al momento de la lectura, deberemos hacer la conversión correspondiente.

Veamos con un ejemplo sencillo cómo utilizar el estado de la aplicación. En el siguiente fragmento de código, guardamos en el estado de la aplicación la fecha de la última visita al sitio, y en la siguiente línea, leemos el valor de la cantidad de visitas contabilizadas hasta el momento:

```
Application["UltimaVisita"] = DateTime.Now;
int visitas = (int)Application
["TotalDeVisitas"];
```


Como podemos notar, estamos almacenando una fecha bajo el nombre UltimaVisita, y recuperando el valor que fue guardado anteriormente con el nombre TotalDeVisitas, al convertirlo de Object a int.

Se nos presenta entonces un problema relacionado con la concurrencia. Por ejemplo, si un usuario necesita hacer varias operaciones seguidas sobre el estado de la aplicación, y otro está ejecutando la misma porción de código, se podría llegar a valores inconsistentes, con el consiguiente resultado no deseado. Para evitar este tipo de conflictos, bloqueamos el estado de la aplicación y lo desbloqueamos al finalizar. Cuando otro usuario quiera acceder al estado de la aplicación, ASP.NET lo dejará en espera hasta que el primera la libere. La clase HttpApplicationState posee un par de métodos llamados Lock() y UnLock() para bloquear y desbloquear, respectivamente:

```
Application.Lock();
Application["UltimaVisita"] = DateTime.Now;
int visitas = (int)Application
["TotalDeVisitas"];
Application.UnLock();
```

El estado de la sesión

Este estado es muy útil para guardar valores que son propios de la aplicación en sí y, por lo tanto, comunes a todos los usuarios. En ocasiones, deberemos mantener el estado de valores propios de cada usuario (por ejemplo, los productos que tiene en su carrito de compras), lo que llamamos "estado de la sesión". Para hacerlo, la clase Page cuenta con la propiedad Session, de tipo HttpSessionState, que funciona de manera muy similar a Application. Es una colección de pares nombre-valor, pero con la diferencia de que la Session de un usuario es totalmente independiente de la de los demás. La forma de utilizarla es similar al uso de Application, con la diferencia de que no deberemos hacer el Lock porque no compartimos la sesión con nadie:

```
Session["Carrito"] = carrito;
Carrito carrito = Session["Carrito"] as
Carrito;
```

Caché

Muchas veces necesitamos guardar del lado del servidor información común a todos los usuarios, costosa de conseguir y que puede cambiar

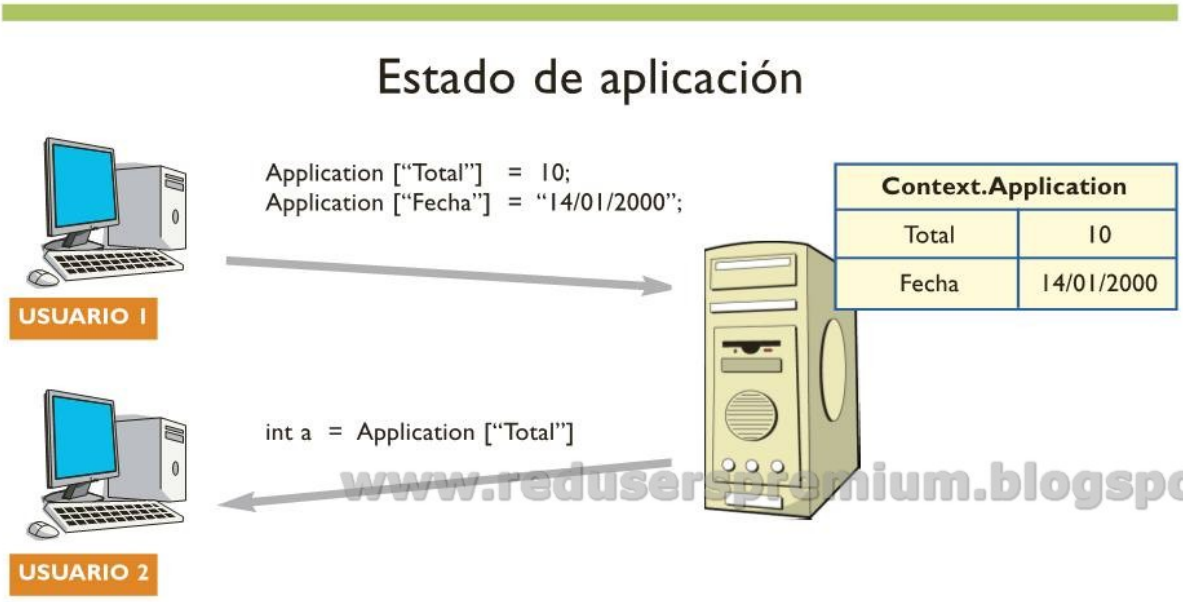


FIGURA 015 | El estado de aplicación es compartido por todos los usuarios.

Los elementos de la caché son eliminados por ASP.NET cuando se cumple la política de expiración.

con el paso del tiempo. Para estos casos, ASP.NET ofrece un espacio de memoria llamado caché. También es una colección de elementos de tipo nombre-valor, pero a diferencia del estado de la aplicación y de la sesión, permite definir políticas de ciclo de vida de los elementos que contiene. El mismo motor de ASP.NET se encarga de gestionarlo, eliminando aquellos elementos cuyo tiempo de duración haya expirado.
Veamos un ejemplo acerca de cómo agregar un elemento a la caché:

```
Cache.Add(
        "provincias",
        listaProvincias,
        null,
        System.Web.Caching.Cache.
        NoAbsoluteExpiration,
        TimeSpan.FromSeconds(30),
        CacheItemPriority.Default,
        null );
```

En este caso, estamos agregando el objeto listaProvincias bajo el nombre o clave "provincias". Aquí el parámetro más importante es el quinto, con el cual le estamos indicando a ASP.NET que dentro de 30 segundos la información habrá expirado.

Global.asax

En ocasiones, necesitamos ejecutar alguna porción de código para hacer alguna inicialización, y colocar valores, ya sea en el estado de la aplicación o en el de la sesión. En estos casos, podemos capturar unos eventos especiales que dispara ASP.NET tanto cuando inicia y termina la aplicación, como cuando

Estado de sesión

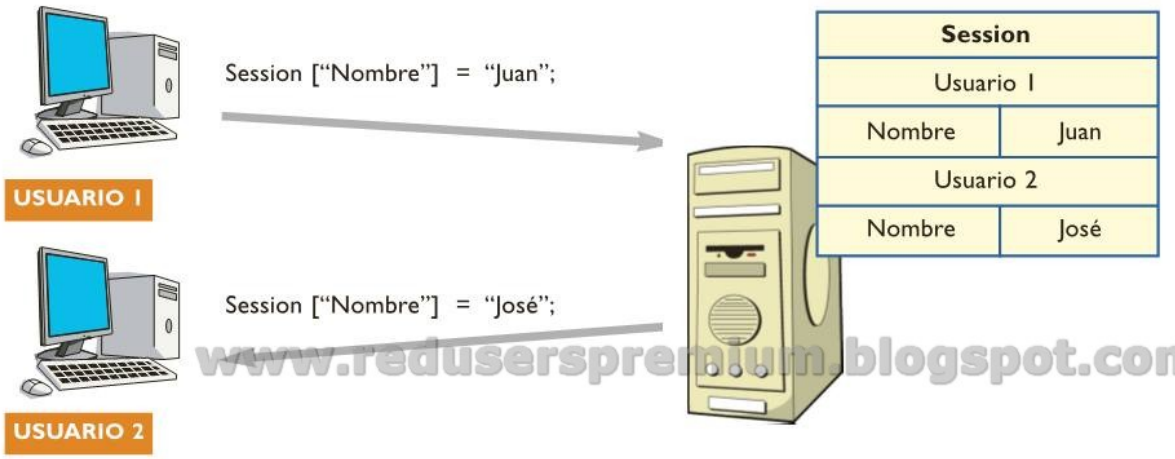


FIGURA 016 | El estado de sesión es independiente para cada usuario.

inicia y termina la sesión. Los manejadores de estos eventos se escriben en el archivo llamado Global.asax.
Si bien hay más eventos que podemos capturar en este archivo, los cuatro más importantes se mencionan en la Tabla 4.

Los eventos Application_Start y Application_End se disparan sólo una vez durante la vida de la aplicación. El primero lo hace cuando el servicio de IIS es iniciado, mientras que el segundo lo hace cuando se detiene el servicio o se apaga el servidor.

Almacenamiento del lado del cliente

Así como podemos almacenar datos del lado del servidor, también podemos hacerlo del lado del cliente, ya sea en su disco rígido o en el texto de la página que viajará hasta el browser. Tenemos dos alternativas principales: las conocidas cookies y el ViewState que usa ASP.NET para mantener el estado de los controles entre los diferentes postbacks.

ViewState

Es una técnica que emplea ASP.NET para guardar el estado de los controles de la página, como los valores de un textbox y/o combobox, y mantenerlos entre cada viaje de ida y vuelta al servidor. Además de almacenar el estado de los controles, ASP.NET permite guardar información que queramos mantener en el cliente. Para hacerlo, contamos con la propiedad de la clase Page, llamada, precisamente, ViewState; ésta, al igual que Application y Session, es una colección de pares nombre-valor:

```
ViewState["clave"] = valor;
```

Si bien puede ser útil guardar información en ViewState, existen dos desventajas fundamentales. Primero, los objetos que queramos almacenar deberán ser serializables, por lo que no podremos guardar cualquier cosa a menos que, de antemano, la hayamos hecho de este tipo. Segundo, el ViewState se renderiza como un

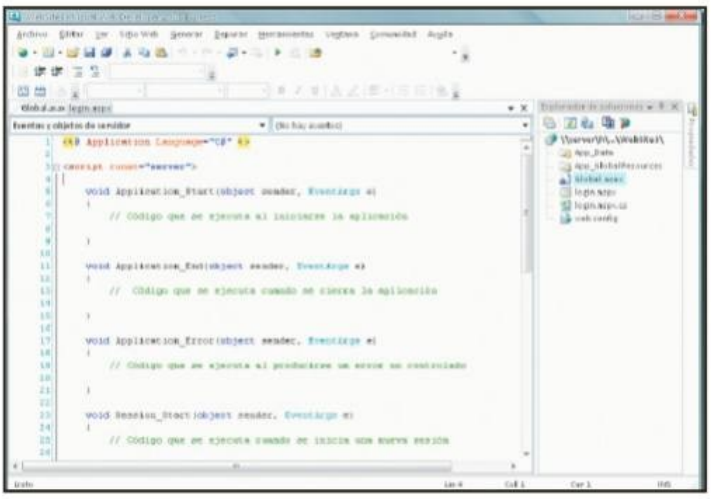


FIGURA 017 | En el archivo Global.asax podemos escribir manejadores para los eventos a nivel de aplicación y sesión.

Tabla 4 | Elementos más importantes de global.asax

Application_Start	Cuando la aplicación es iniciada.
Application_End	Cuando la aplicación es finalizada.
Session_Start	Cuando un usuario realiza el primer pedido a la aplicación.
Session_End	Cuando se eliminan de la memoria los datos de la sesión creada por el usuario.

HAY DOS ALTERNATIVAS AL MOMENTO DE ALMACENAR DATOS DEL LADO DEL CLIENTE: LAS CONOCIDAS COOKIES Y EL VIEWSTATE QUE USA ASP.NET PARA MANTENER EL ESTADO DE LOS CONTROLES ENTRE LOS DIFERENTES POSTBACKS.

campo oculto en el texto de la página, y esto debe viajar al cliente; por lo tanto, si no tenemos cuidado con el tamaño de los valores que guardamos, la página puede demorar mucho en llegar al cliente y ser mostrada por el browser.

Cookies

La otra alternativa de la que disponemos es el uso de las cookies, que no son parte de ASP.NET en sí sino del protocolo HTTP. En una cookie podemos guardar pequeños valores, ya sea en el disco del cliente o en la memoria (cookies transitorias). El uso de cookies no se recomienda a menos que sea extremadamente necesario, por dos motivos: no fueron diseñadas para almacenar grandes valores y la mayoría de los browsers brindan la opción de deshabilitar el uso de cookies por razones de seguridad, en cuyo caso la aplicación no mantendrá los valores que cree mantener en el cliente.

La forma de acceder a las cookies es mediante la propiedad Cookies, de Response, de la clase Page:

```
Response.Cookies.Add(
        new HttpCookie("FechaUltimaVisita",
        DateTime.Now.ToString())
);
```

FIGURA 018 | El ViewState se almacena en el texto de la página, como un campo oculto.

Argentina $8,90 (recargo al interior $0,20)) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net 9

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

ASP.NET Avanzado

Configuración - Autenticación
Páginas maestras

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Administrar el estado

Global.asax - ViewState
Sesión - Cookies

ISBN 978-987-1347-43-8
9 789871 347438 00009