

Introducción a ASP.NET

Francisco Ortega Belmonte

PID_00194034



Los textos e imágenes publicados en esta obra están sujetos –excepto que se indique lo contrario– a una licencia de Reconocimiento-Compartir igual (BY-SA) v.3.0 España de Creative Commons. Se puede modificar la obra, reproducirla, distribuirla o comunicarla públicamente siempre que se cite el autor y la fuente (FUOC. Fundació per a la Universitat Oberta de Catalunya), y siempre que la obra derivada quede sujeta a la misma licencia que el material original. La licencia completa se puede consultar en: <http://creativecommons.org/licenses/by-sa/3.0/es/legalcode.ca>

Índice

Introducción	5
Objetivos	6
1. El entorno de desarrollo “.NET Framework”	7
1.1. C# y Visual Basic	7
1.2. El Common Language Runtime (CLR)	8
1.3. La biblioteca de clases de .NET	8
2. El lenguaje de programación C#	9
2.1. Características y peculiaridades	9
2.2. Variables y tipos de datos	10
2.2.1. Asignaciones e inicializaciones	11
2.2.2. <i>Arrays</i>	11
2.2.3. <i>ArrayList</i>	12
2.2.4. Enumeraciones	12
2.2.5. Operaciones con variables	12
2.2.6. Conversiones de tipo	13
2.2.7. El tipo <i>String</i>	14
2.2.8. Los tipos <i>DateTime</i> y <i>TimeSpan</i>	15
2.3. Condicionales lógicos	16
2.4. Bucles	17
2.5. Métodos	18
2.6. Clases y objetos	19
3. Introducción a las herramientas de desarrollo	22
3.1. Visual Studio	22
3.1.1. Instalación de Visual Web Developer	24
3.1.2. Galería de plantillas	25
3.1.3. Explorador de soluciones	26
3.1.4. El diseñador de páginas	27
3.1.5. El editor de código C#	29
3.1.6. Ejecución y depuración de un sitio web	31
3.1.7. Base de datos compacta con Visual Studio	34
4. Introducción a ASP.NET	45
4.1. Características principales de ASP.NET	45
4.2. La evolución de ASP.NET	47
4.2.1. ASP.NET 1.0 y 1.1	47
4.2.2. ASP.NET 2.0	47
4.2.3. ASP.NET 3.5	48

4.2.4.	ASP.NET 4.0	48
4.3.	Estructura de una aplicación ASP.NET	49
4.3.1.	Tipos de archivos en ASP.NET	50
4.4.	¿Qué són los WebForm?	51
4.4.1.	Añadiendo un WebForm a nuestro sitio web	52
4.5.	La clase Page.....	53
4.6.	El ViewState	56
4.7.	Controles de servidor	58
4.7.1.	Controles de servidor HTML	59
4.7.2.	Controles web	61
4.7.3.	Controles Rich	64
4.7.4.	Controles de validación	66
4.7.5.	Controles de datos	68
4.8.	Controles de usuario	71
4.8.1.	Creando un UserControl	72
4.9.	Páginas maestras	74
4.9.1.	Componentes ContentPlaceholder y Content.....	75
4.9.2.	Ejemplo práctico	76
5.	ASP.NET AJAX.....	81
5.1.	El ScriptManager	81
5.2.	Renderizaciones parciales con UpdatePanel	82
5.3.	Indicación de estado con UpdateProgress	85
5.4.	ASP.NET AJAX Control Toolkit	87
5.4.1.	Instalando ASP.NET AJAX Control Toolkit	87
5.4.2.	Accordion	91
5.4.3.	AutoCompleteExtender	93

Introducción

ASP.NET es una plataforma de desarrollo de Microsoft de las más extendidas en la programación web. Combinado con el entorno de desarrollo por excelencia de Microsoft, Visual Studio 2010, se pueden crear sofisticadas aplicaciones web de forma visual y rápida.

En este módulo se ofrece una introducción al entorno de desarrollo (*framework*) .NET, una descripción a grandes rasgos del lenguaje de programación C# y, por último, una introducción a la plataforma de desarrollo ASP.NET 4.0 incluidos los controles ASP.NET AJAX, que es uno de los aspectos diferenciadores respecto a otras plataformas de desarrollo.

Objetivos

Los objetivos de este módulo didáctico son los siguientes:

- 1.** Mostrar el uso de tecnología AJAX en uno de los entornos de desarrollo más utilizados hoy en día: .NET.
- 2.** Conocer qué es ASP.NET y su implementación AJAX.
- 3.** Familiarizarse con el entorno de desarrollo Visual Studio 2010.
- 4.** Crear sitios web y diseñar páginas ASP.NET interactivas que respondan a las diferentes acciones de los usuarios.
- 5.** Comprender el ciclo de vida de una página ASP.NET entre diferentes tipos de peticiones.

1. El entorno de desarrollo “.NET Framework”

.NET es un término utilizado por Microsoft para dar nombre a un conjunto de tecnologías –algunas revolucionarias, otras no tanto– que están diseñadas para ayudar a los desarrolladores a crear diferentes tipos de aplicaciones, desde aplicaciones de escritorio o servicios de Windows hasta aplicaciones web.

El conjunto de tecnologías que forman el entorno de desarrollo¹ “.NET Framework” son las siguientes:

⁽¹⁾En inglés, *framework*.

- Los **lenguajes de programación .NET**: entre ellos Visual Basic, C#, F# y C++.
- El **Common Language Runtime (CLR)**: es el motor que ejecuta todos los programas de .NET y que provee servicios automáticos para estas aplicaciones, como la comprobación de seguridad o la gestión de memoria.
- La **biblioteca de clases de .NET**: contiene miles de clases, interfaces, estructuras, ... con una funcionalidad predefinida que se pueden utilizar en las aplicaciones .NET.
- **ASP.NET**: es el motor que permite ejecutar las aplicaciones web creadas en .NET.
- **Visual Studio**: es la herramienta de desarrollo por excelencia de Microsoft. Permite realizar cualquier tipo de aplicación .NET y su depuración.

1.1. C# y Visual Basic

Durante este apartado, como se podrá ver, utilizaremos el lenguaje C# para realizar los ejemplos. Este lenguaje se asemeja bastante a Java y a C++. A su vez C# y Visual Basic son bastante similares; de hecho, casi cualquier bloque de código puede ser traducido de un lenguaje a otro, pero su sintaxis es diferente.

Tanto C# como Visual Basic utilizan la biblioteca de clases de .NET y son compatibles con el CLR.

1.2. El Common Language Runtime (CLR)

El Common Language Runtime (CLR²) es el motor que soporta todos los lenguajes .NET. En realidad, soporta el lenguaje intermedio entre el lenguaje nativo y el lenguaje de programación Microsoft Intermediate Language (MSIL³). El CLR interpreta este código intermedio y lo transforma en código máquina.

⁽²⁾CLR es la sigla de Common Language Runtime.

⁽³⁾MSIL es la abreviatura de Microsoft Intermediate Language.

Para entenderlo mejor, el CLR sería el símil en Microsoft de la máquina virtual en Java.

1.3. La biblioteca de clases de .NET

La biblioteca de clases de .NET es un repositorio gigante de clases que proporcionan una funcionalidad predefinida, desde la lectura de un archivo XML hasta el envío de un correo electrónico. Cualquiera de los lenguajes .NET puede utilizar dicha biblioteca de clases.

Ciertas partes de la biblioteca nunca se utilizan en programación web y solo se utilizan para realizar aplicaciones de escritorio, mientras que otras están concebidas exclusivamente para aplicaciones web.

Como hemos podido ver, la filosofía de Microsoft es proporcionar la infraestructura de las funcionalidades más comunes para que los desarrolladores solo tengan que programar código específico de la aplicación.

2. El lenguaje de programación C#

Antes de empezar a crear una aplicación ASP.NET, el programador debe elegir el lenguaje de programación, C# o Visual Basic. La opción normal si no se ha programado nunca es Visual Basic por parecerse más al lenguaje natural; en cambio si ya se ha programado antes en C, C++ o Java, la elección suele ser C#.

En este módulo mostraremos las características principales de C#, los tipos de datos, las operaciones, los condicionales o los bucles existentes.

2.1. Características y peculiaridades

Las principales características que definen C# como uno de los lenguajes más potentes y modernos de la actualidad son:

- a) **Es orientado a objetos.** A excepción de algunos datos simples como los enteros, flotantes o *strings*, el resto de las entidades son clases predefinidas en las bibliotecas de clases de .NET.
- b) **Es un lenguaje compilado.** Se necesita un compilador específico que traduzca el lenguaje C# a código MSIL, que, como sabemos, es el que interpreta el CLR.
- c) **Dispone de la biblioteca de clases .NET.** Desde C# tenemos acceso a toda la biblioteca de clases de .NET, que implementa multitud de funcionalidades.
- d) **Dispone de *garbage collector*.** La destrucción y liberación de memoria se gestiona de forma automática totalmente transparente al desarrollador.

Por otro lado, alguna de las peculiaridades que diferencian a C# de otros lenguajes de programación son:

- a) **Diferenciación entre mayúsculas y minúsculas.** En C# no es lo mismo definir una variable llamada *prueba* a otra variable *Prueba*. Algunos lenguajes de programación, como por ejemplo Visual Basic, no hacen esta distinción.
- b) **Comentarios.** Los comentarios son textos descriptivos que son ignorados por el compilador. C# dispone de dos tipos de comentarios básicos:

- El comentario de una línea:

```
// Comentario de una línea
```

- El comentario de múltiples líneas:

```
/* Comentario de multiples
lineas */
```

c) **Terminación de sentencia.** Toda sentencia en C# debe terminar con el carácter “;”. Es importante recordar que una sentencia puede componerse de más de una línea, como podemos ver a continuación:

```
// Sentencia en una línea
valor = valor1 + valor2 + valor3;
// Sentencia en varias líneas
valor = valor1 + valor2 +
valor3;
```

d) **Bloques.** Todo conjunto de sentencias debe empezar con el carácter “{” y terminar por el carácter “}”. Estos conjuntos pueden ser funciones, clases, bucles, etc.

```
{
//Conjunto de sentencias
}
```

2.2. Variables y tipos de datos

Al igual que todos los lenguajes de programación en C#, se pueden mantener datos en memoria utilizando variables. Las variables pueden almacenar números, texto, fechas, horas e incluso pueden guardar direcciones a otros objetos. Cuando se declara una variable se le asigna un nombre y se especifica el tipo de datos al que pertenece.

```
//Declaración de un entero llamado código
int código;
//Declaración de un string llamado nombre
string nombre;
```

En la tabla siguiente podemos ver los tipos de datos disponibles en C#:

Figura 1. Tipos de datos en C#

Nombre del tipo	Descripción
byte	Entero de 0 a 255
short	Entero de -32,768 a 32,767
int	Entero de -2,147,483,648 a 2,147,483,647
long	Entero de -9.2e18 a 9.2e18
float	Número de punto flotante de -3.4e38 a 3.4e38

Nombre del tipo	Descripción
double	Número de punto flotante de $-1.8e308$ a $1.8e308$
decimal	Número fraccionario de 128-bit
char	Carácter Unicode
string	Conjunto de caracteres Unicode
bool	Valor de <i>true</i> o <i>false</i>
DateTime	Representación de fechas

2.2.1. Asignaciones e inicializaciones

Cuando se declara una variable, se le pueden asignar valores siempre y cuando estos valores sean del tipo correcto.

```
//Declaración de variables
int código;
string nombre;
//Asignación de valores
codigo = 10;
nombre = "UOC"
```

También se puede asignar un valor en el mismo momento en que se declara la variable.

```
int codigo = 10;
string nombre = "UOC"
```

2.2.2. Arrays

Los *arrays* permiten al desarrollador almacenar una serie de valores del mismo tipo. Puede accederse a cada valor mediante la posición que ocupa en el *array*, teniendo en cuenta que el primer valor de un *array* tiene índice 0.

```
//Creación de un array con cuatro strings
string[] stringArray = new string[4];
//Creación e inicialización de un array con cuatro strings
string[] stringArray = {"1","2","3","4"};
```

Para acceder a un elemento, se ha de especificar el índice entre corchetes [].

```
int[] intArray = {1,2,3,4};
int elemento = intArray[2]; //elemento se inicializa con un 3
```

2.2.3. ArrayList

Los *arrays* en C# no pueden ser redimensionados, es decir, una vez creados no se pueden cambiar de tamaño. Si se necesita un *array* dinámico, se puede utilizar una clase de la biblioteca de clases de .NET llamada *ArrayList*, que además acepta que los elementos de los *arrays* sean de cualquier tipo.

Utilización de la clase ArrayList

A continuación vemos un ejemplo de utilización de la clase *ArrayList*:

```
//Creación de un objeto ArrayList
ArrayList listaDinamica = new ArrayList();
//Adición de diferentes strings a la lista
listaDinamica.Add("primero");
listaDinamica.Add("segundo");
listaDinamica.Add("tercero");
```

2.2.4. Enumeraciones

Una enumeración es un grupo de constantes relacionadas, a cada una de las cuales se le da un nombre descriptivo. Cada valor de una enumeración corresponde a un número entero prefijado.

```
//Definición de una enumeración llamada TipoUsuario
enum TipoUsuario
{
    Admin,
    Invitado,
    Erroneo
}
```

Ahora ya se puede utilizar *TipoUsuario* como un tipo de datos especial con solo tres valores.

```
//Creación de un nuevo valor de tipo Admin
TipoUsuario usuario = TipoUsuario.Admin;
```

2.2.5. Operaciones con variables

En C# el desarrollador puede utilizar todos los tipos estándar de operaciones. Cuando se trabaja con tipos numéricos, se pueden utilizar los símbolos matemáticos que podemos ver en la tabla siguiente:

Figura 2. Operaciones en C#

Operador	Descripción	Ejemplo
+	Suma	1 + 1 = 2
-	Resta	5 - 2 = 3
*	Multiplicación	2 * 5 = 10

Operador	Descripción	Ejemplo
/	División	5.0 / 2 = 2.5
%	Resto de la división entera	7 % 3 = 1

Para controlar el orden en que se ejecutan, las expresiones pueden agruparse entre paréntesis.

```
int numero;
numero = 4 + 2 * 3;
//numero será 10
numero = (4 + 2) * 3
//numero será 18
```

La división en C# puede ser algo confusa. Si se divide un número entero por otro entero, C# realiza una división entera, es decir, se descarta automáticamente la parte fraccionaria del resultado. La solución a este problema es definir uno de los números, o ambos, como fraccionarios; por ejemplo, 5 pasaría a ser 5.0. De esta manera obtendremos un resultado correcto.

En C# la operación adición + se puede utilizar también para concatenar *strings*.

```
//Concatenación de tres strings
nombreCompleto = nombre + " " + apellidos;
```

En C# cuando la operación y la asignación se hacen sobre la misma variable se pueden utilizar los operadores de la siguiente forma:

```
//Añadir 10 a valor. Es lo mismo que valor = valor + 10
valor += 10
//Multiplicar valor por 3. Es lo mismo que valor = valor * 3
valor *= 3
```

2.2.6. Conversiones de tipo

Convertir información de un tipo de dato a otro es algo bastante común en el mundo de la programación. Las conversiones de datos pueden ser de dos tipos: implícitas o explícitas; por ejemplo, la conversión de un tipo `int` a un `long` es implícita, porque no implica pérdida de información; sin embargo, la conversión de un tipo `long` a un `int` es explícita debido a que en esta puede haber pérdida de información.

Para las conversiones implícitas no se necesita ningún código especial:

```
int valorPequeno;
long valorGrande;
//La conversión siempre se puede realizar porque un long puede contener a un int
```

```
valorGrande = valorPequeno;
```

En cambio, en las conversiones explícitas, es necesario indicar el tipo entre paréntesis a la izquierda de la variable.

```
int valorGrande = 1000;
short valorPequeno;
//Si el valorGrande supera la capacidad de un short habrá pérdida de información.
valorPequeno = (short)valorGrande;
```

2.2.7. El tipo String

La clase `String` de la biblioteca de clases de .NET nos permite manipular de múltiples maneras una cadena de texto.

Ejemplo

```
string mistring = "Esto es un string de ejemplo ";
mistring = mistring.Trim();           //"Esto es un string de ejemplo";
mistring = mistring.Substring(0,4);   //"Esto";
mistring = mistring.ToUpper();        //"ESTO";
mistring = mistring.Replace("O","A"); //"ESTA";

int longitud = mistring.Length;       // = 4;
```

En los ejemplos anteriores cada método de la clase `String` genera un nuevo *string* que es asignado a la variable `mistring`. Todas estas sentencias se pueden realizar en una sola línea:

```
mistring = mistring.Trim().Substring(0,4).Replace("O","A");
```

En la tabla siguiente vemos alguno de los métodos y propiedades más importantes de la clase `String`:

Figura 3. Funciones y propiedades de la clase `String`

Miembro	Descripción
<code>Length</code>	Retorna el número de caracteres del <i>string</i> .
<code>ToUpper()</code> y <code>ToLower()</code>	Retorna un <i>string</i> con todos los caracteres del <i>string</i> original en mayúscula o minúscula.
<code>Trim()</code>	Retorna un <i>string</i> sin los espacios del <i>string</i> original.
<code>Insert()</code>	Inserta otro <i>string</i> en el <i>string</i> original en la posición indicada.
<code>Remove()</code>	Elimina un número determinado de caracteres desde una posición determinada.
<code>Replace()</code>	Reemplaza un <i>substring</i> del <i>string</i> original con otro <i>string</i> .
<code>Substring()</code>	Devuelve una porción del <i>string</i> de una longitud determinada.
<code>StartsWith()</code> y <code>EndsWith()</code>	Determina si un <i>string</i> comienza o acaba en un determinado <i>substring</i> .

Miembro	Descripción
Split()	Devuelve un <i>array</i> con todos los <i>substrings</i> resultantes de seccionar el <i>string</i> por un carácter determinado.
Join()	Fusiona un <i>array</i> de <i>strings</i> en un <i>string</i> único.

2.2.8. Los tipos `DateTime` y `TimeSpan`

Los tipos de datos `DateTime` y `TimeSpan` disponen de métodos y propiedades para manipular fechas y tiempos respectivamente.

Ejemplos

Podemos ver a continuación un ejemplo de manipulación de fechas con `DateTime`:

```
DateTime miFecha = DateTime.Now;
miFecha = miFecha.AddDays(100);
string dateString = miFecha.Year.ToString();
```

En el ejemplo siguiente vemos cómo buscar la diferencia en minutos entre dos fechas:

```
DateTime miFecha1 = DateTime.Now;
DateTime miFecha2 = DateTime.Now.AddHours(3000);
TimeSpan diferencia;
diferencia = miFecha2.Subtract(miFecha1);
double numeroMinutos;
numeroMinutos = diferencia.TotalMinutes;
```

Las clases `DateTime` y `TimeSpan` también soporta los operadores aritméticos `+` y `-`, incluso entre un objeto `DateTime` y otro `TimeSpan`.

Ejemplo

```
DateTime miFecha1 = DateTime.Now;
TimeSpan intervalo = TimeSpan.FromHours(3000);
DateTime miFecha2 = miFecha1 + intervalo;
```

A continuación podemos ver dos tablas con algunos métodos y propiedades de las clases `DateTime` y `TimeSpan`:

Figura 4. Funciones y propiedades de la clase `DateTime`

Miembro	Descripción
Now	Devuelve la fecha y hora actual.
Today	Devuelve la fecha actual.
Year, Date, Month, Hour, Minute, Second y Millisecond	Devuelve una parte de un <code>DateTime</code> .
DayOfWeek	Devuelve el día de la semana de un <code>DateTime</code> .
Add() y Subtract()	Añade o resta un <code>TimeSpan</code> de un <code>DateTime</code> .
AddYears(), AddMonths(), AddDays(), AddHours(), AddMinutes(), AddSeconds(), AddMilliseconds()	Añade una porción de tiempo a un <code>DateTime</code> .
DaysInMonth()	Devuelve el número de día de un determinado mes.

Figura 5. Funciones y propiedades de la clase `TimeSpan`

Miembro	Descripción
<code>Days</code> , <code>Hours</code> , <code>Minutes</code> , <code>Seconds</code> , <code>Milliseconds</code>	Devuelve una parte de un <code>TimeSpan</code> .
<code>TotalDays</code> , <code>TotalHours</code> , <code>TotalMinutes</code> , <code>TotalSeconds</code> , <code>TotalMilliseconds</code>	Devuelve el total de días, horas,... que tiene un <code>TimeSpan</code> .
<code>Add()</code> y <code>Subtract()</code>	Añade o resta un <code>TimeSpan</code> a otro <code>TimeSpan</code> .
<code>FromDays()</code> , <code>FromHours()</code> , <code>FromMinutes()</code> , <code>FromSeconds()</code> , <code>FromMilliseconds()</code>	Construye un <code>TimeSpan</code> a partir de días, horas, etc.

2.3. Condicionales lógicos

En muchas ocasiones se necesita una condición lógica para decidir qué acción tomar sobre la base de una información determinada. Una condición lógica puede ser evaluada como verdadera o falsa y en función de esto, ejecutar un bloque de código u otro. Para crear una condición podemos utilizar cualquier combinación de variables o literales con los operadores lógicos que vemos en la siguiente tabla:

Figura 6. Condicionales en C#

Operador	Descripción
<code>==</code>	Igual a
<code>!=</code>	Diferente de
<code><</code>	Menor que
<code>></code>	Mayor que
<code><=</code>	Menor o igual a
<code>>=</code>	Mayor o igual a
<code>&&</code>	AND lógico
<code> </code>	OR lógico

La sentencia `if` es la pieza clave de la lógica condicional en la programación. A continuación vemos un ejemplo donde se evalúa una condición y se ejecuta un bloque de código u otro:

```
if (numero > 10)
{
    //Bloque A
}
else
{
    //Bloque B
}
```


C# también pone a nuestra disposición la sentencia `switch`, que nos sirve para evaluar una variable para varios posibles valores. La única limitación de la sentencia `switch` es que la variable solo puede ser de los tipos `int`, `bool`, `char`, `string` o el valor de una enumeración.

```
switch (numero)
{
    case 1:
        //Bloque A
        break;
    case 1:
        //Bloque B
        break;
    default:
        //Bloque C
        break;
}
```

Vemos que en cada bloque de código debemos indicar la palabra `break`; si no lo hiciéramos, se seguiría ejecutando el código de forma secuencial.

2.4. Bucles

En C#, al igual que en la mayoría de lenguajes, existen tres tipos de bucles para ejecutar bloques de código de forma iterativa: `for`, `foreach` y `while`;

El bucle `for` permite al programador poder repetir un bloque de código un número predeterminado de veces. Para crear un `for` es necesario especificar un valor inicial, un valor final y el incremento de cada iteración.

```
for (int i = 0; i < 10; i++)
{
    //Bloque de código
}
```

El bucle `foreach` nos permite recorrer los elementos de un conjunto de datos. Con un `foreach` no es necesario crear una variable para el contador y además es especialmente útil para recorrer listas, *arrays* o colecciones de datos en general.

```
string[] stringArray = {"uno","dos","tres"};
foreach (string element in stringArray)
{
    System.Diagnostics.Debug.Write(element + " ");
}
```

Finalmente, C# dispone del bucle `while` que sirve para comprobar una condición específica antes o después de cada iteración de un bloque de código. Cuando esta condición es falsa, se sale del bucle; por el contrario, mientras sea verdadera se van realizando iteraciones en el bloque de código.

```
int i = 0;
while (i < 10)
{
    i += 1;
}
```

Otra forma de implementar el bucle `while` es poniendo la condición al final del bloque de código; de esta forma nos aseguramos de que el código se ejecuta por lo menos una vez.

```
int i = 0;
do
{
    i += 1;
} while (i < 10);
```

2.5. Métodos

Los métodos en C# son el bloque de código más básico que podemos utilizar para organizar nuestro código. Cuando se declara un método, la primera parte de la declaración indica el tipo de datos del valor de retorno y la segunda el nombre de método.

```
//Método que no devuelve ningún valor
void MetodoQueNoDevuelveValor()
{
    //Código
}

//Método que retorna un entero
int MetodoQueRetornaUnInt()
{
    return 10;
}
```

Los métodos pueden aceptar parámetros. Los parámetros se declaran de forma muy similar a la declaración de variables.

```
int sumaNumeros (int numero1,int numero2)
{
    return numero1 + numero2;
}
```

```
//Llamada al método
int resultado = sumaNumeros(10,10);
```

Cabe destacar que en la declaración de los parámetros podemos darles un valor por defecto.

```
private string NombreUsuario (int ID, bool esAdmin = false)
{
    //Código
}
```

Una característica propia de C# relacionado con los métodos es la sobrecarga, es decir, podemos crear más de un método con el mismo nombre pero con diferentes parámetros.

```
public decimal GetPrecioProducto(int ID)
{
    //Código
}
public decimal GetPrecioProducto(string nombre)
{
    //Código
}
```

2.6. Clases y objetos

C#, como hemos dicho, es un lenguaje de programación orientado a objetos. Una clase es la definición del código de un objeto y la creación de una clase en C# se realiza de la siguiente forma:

```
public class MiClase
{
    //Código de la clase
}
```

En una clase podemos definir variables que sean variables miembro o propiedades de la clase. Estas variables pueden tener diferentes niveles de acceso, que se especifican en la declaración de la variable.

```
public class Producto
{
    private string nombre ;
    private decimal precio;
}
```

En la siguiente tabla podemos ver los diferentes niveles de acceso disponibles en C#:

Figura 7. Niveles de acceso en C#

Nivel de acceso	Accesibilidad
public	Accesible por cualquier clase
private	Solo accesible por miembros de la propia clase
internal	Accesible por cualquier clase del ensamblado actual
protected	Accesible por miembros de la propia clase o por clases que hereden de esta

Para una clase también podemos definir una serie de métodos que llamaremos *métodos miembro*, y al igual que una variable miembro, deben tener el nivel de acceso definido en la declaración.

```
public class Producto
{
    private string nombre ;
    private decimal precio;

    public decimal GetPrecio()
    {
        return this.precio;
    }
}
```

Un método especial que todas las clases deben tener es el constructor de clase, que es el encargado de inicializar los parámetros de la clase. Puede haber uno o más constructores, cuyo nombre será el mismo de la clase a la que pertenecen, pero que se diferenciarán en el conjunto de los parámetros que se le pase a cada uno.

```
public class Producto
{
    private string nombre ;
    private decimal precio;

    public void Producto(int precioInicio)
    {
        this.precio = precioInicio;
    }

    public decimal GetPrecio()
    {
        return this.precio;
    }
}
```

```
}
```

Una vez hemos construido una clase como la anterior, podemos crear tantas instancias de ella como queramos; cada una de ellas será lo que se denomina un *objeto*.

```
Product productoVenta = new Product(50);  
int precio = productoVenta.GetPrecio();
```

3. Introducción a las herramientas de desarrollo

Las aplicaciones ASP.NET pueden crearse con un simple editor de textos, introduciendo todo el código –tanto el diseño de la interfaz (XHTML/CSS/Javascript) como la lógica (código C#)– en módulos .aspx, que serían compilados dinámicamente por el motor de ASP.NET en el momento de recibir las solicitudes.

Esta, sin embargo, es una solución viable únicamente para proyectos muy pequeños, ya que es difícil mantener un diseño de interfaces a partir de la escritura manual de los atributos de todos los elementos, y de una mezcla de códigos en diferentes lenguajes en un mismo archivo.

Dependiendo de cuáles sean nuestros objetivos y necesidades, tenemos a nuestra disposición diferentes herramientas útiles para el desarrollo de aplicaciones basadas en ASP.NET. El punto de partida sería Visual Web Developer Express. Se trata de una herramienta gratuita, descargable directamente desde el sitio web de Microsoft y con los elementos necesarios para construir aplicaciones básicas y de nivel medio, con limitaciones sobre todo a la hora de trabajar con servidores de bases de datos.

Más sofisticados son los entornos de trabajo que ofrecen las diferentes versiones de Visual Studio 2010, que además de aplicaciones ASP.NET, tienen capacidad para desarrollar muchos tipos de proyectos: aplicaciones para Windows, para dispositivos móviles, para Office, etc.

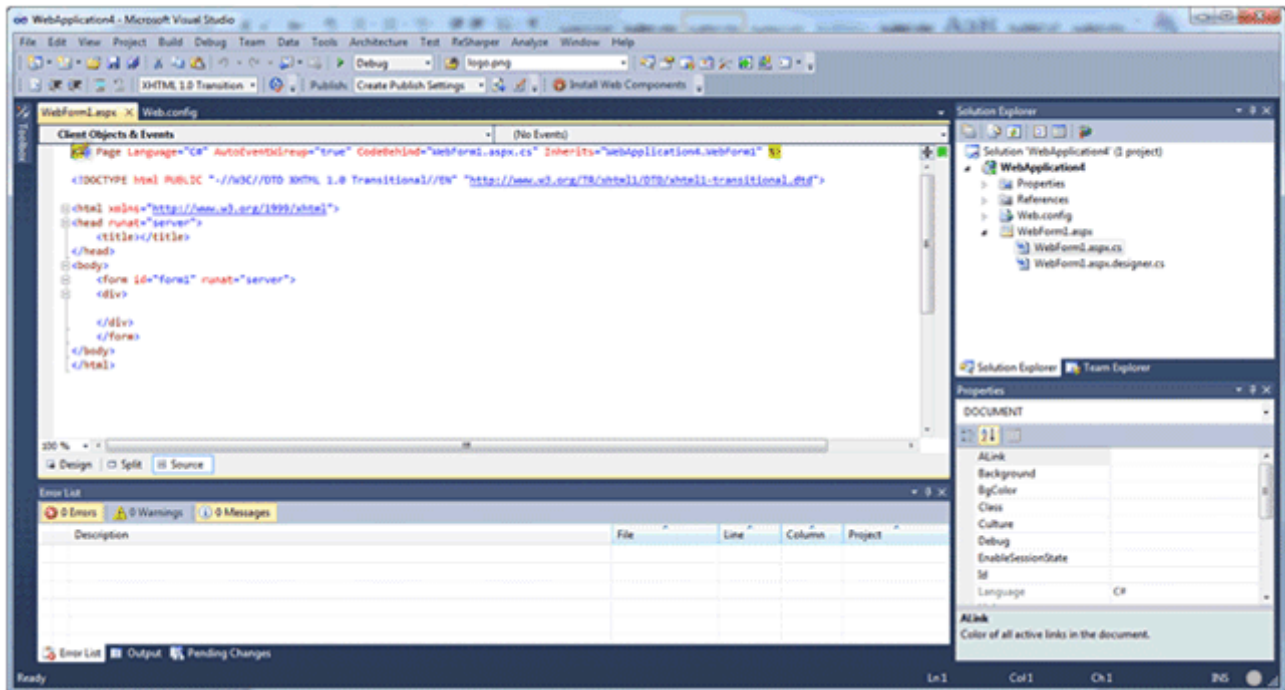
3.1. Visual Studio

Desde hace años, la herramienta de desarrollo por excelencia para plataformas Windows se denomina Microsoft Visual Studio, un producto que reúne compiladores para diferentes lenguajes, editores de código con avanzadas características de ayuda a la codificación, diseñadores de interfaces de usuario, depuración integrada, gestión de proyectos complejos, edición de bases de datos tanto locales como remotas, etc.

La última versión de Visual Studio se apellida 2010 (figura 8), y va unida a la versión 4.0 de la plataforma .NET. A la hora de iniciar un nuevo proyecto, sin embargo, es posible elegir la versión de la plataforma .NET sobre la que se ejecutará: 2.0, 3.0, 3.5 o 4.0. Esto permite usar una única herramienta para el desarrollo de aplicaciones sobre distintas versiones de la plataforma.

Hay disponibles múltiples ediciones de Visual Studio dirigidas tanto a usuarios individuales (Standard y Professional) como a grupos de trabajos (Team System). En lo relativo al desarrollo de aplicaciones web basadas en ASP.NET, sus capacidades son básicamente las mismas.

Figura 8. Visual Studio 2010



A diferencia de Visual Studio, Visual Web Developer es una herramienta específica para el desarrollo de aplicaciones web con ASP.NET que no cuenta con opciones para la construcción de otros tipos de proyectos. Por lo tanto, si necesitamos diseñar soluciones complejas en las que la aplicación web sea solamente una de las partes, Visual Web Developer nos resultará insuficiente y tendremos que recurrir a alguna de las ediciones de Visual Studio. En lo referente a nuestro curso, tendremos más que suficiente con esta herramienta.

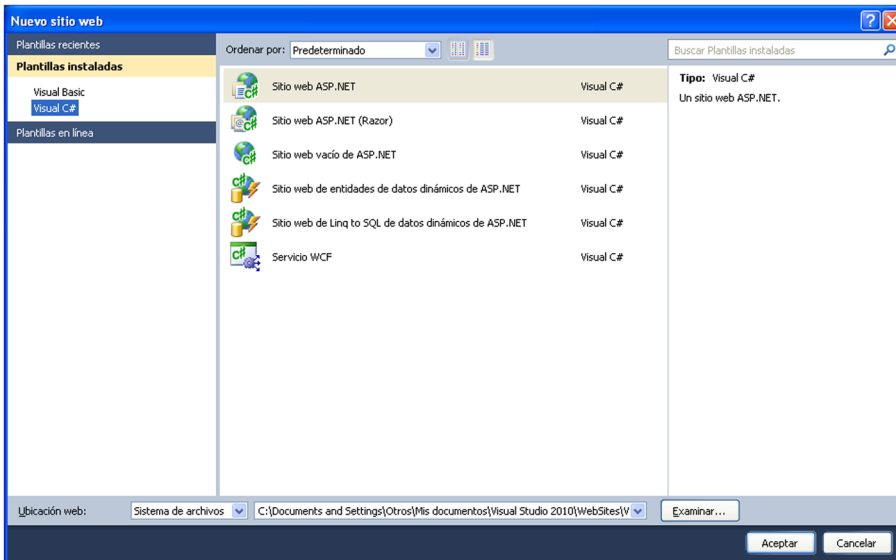
Visual Web Developer permite usar como lenguaje de implementación tanto Visual Basic como C#, pero no permite seleccionar la versión de la plataforma .NET sobre la que se ejecutará el proyecto, asumiendo siempre que será la 4.0.

Otra diferencia respecto a Visual Studio es que Visual Web Developer solamente permite trabajar con bases de datos locales, no remotas. Tampoco existe la posibilidad de efectuar depuración remota con la aplicación ejecutándose en un equipo que no sea el de desarrollo.

Por lo demás, Visual Web Developer incorpora el mismo diseñador de páginas ASP.NET que Visual Studio, el mismo editor de código, los mismos componentes y también los mismos servicios fundamentales: los de la plataforma .NET 4.0.

El gestor de proyectos es prácticamente idéntico al de Visual Studio, al igual que el explorador de bases de datos, el explorador de objetos y muchas otras utilidades incluidas en el entorno.

Figura 9. Formulario de creación con las diferentes plantillas disponibles



3.1.1. Instalación de Visual Web Developer

Para que podamos descubrir las capacidades de Visual Web Developer primero es necesaria la instalación de la herramienta. Para ello accederemos a la página de Microsoft Visual Web Developer 2010 Express.

Una vez en la página, haremos clic sobre el botón *Install now*. Esto lanzará el Web Platform Installer, que nos descargará e instalará el producto automáticamente.

Figura 10. Web Platform Installer



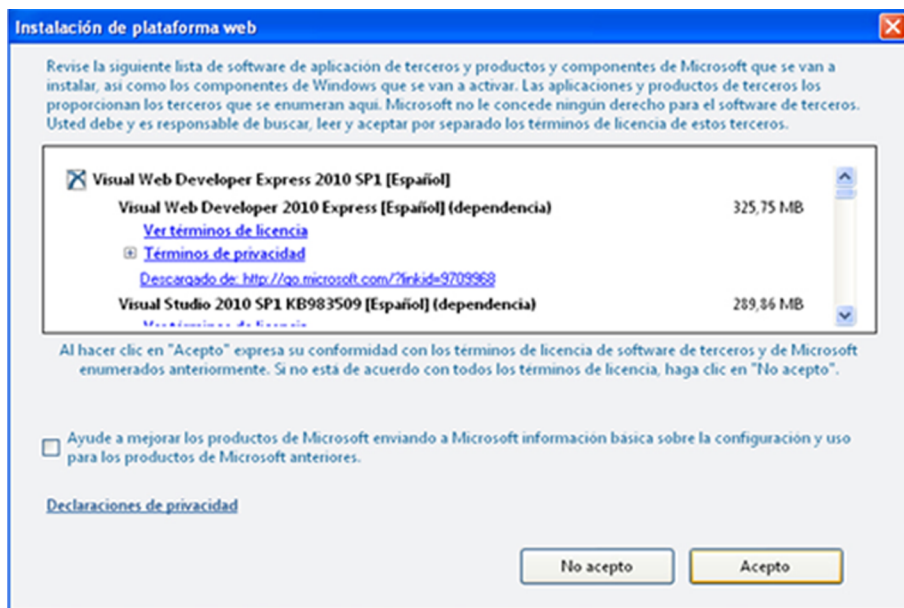
Una vez Web Platform Installer inicie el instalador de Visual Web Developer, pulsamos en el botón *Instalar*.

Figura 11. Instalando Visual Web Developer Express



Aceptamos las condiciones de uso y el programa de instalación descargará Visual Web Developer y lo instalará de forma autónoma.

Figura 12. Condiciones de uso



3.1.2. Galería de plantillas

Al elegir la opción *File > New > Web Site*, o hacer clic en el botón equivalente en la barra de herramientas, vemos la galería de plantillas instaladas. Cada plantilla permite crear un tipo de proyecto diferente o contener una serie de elementos distintos de partida.

En la galería de plantillas (figura 9) siempre elegiremos la opción *Sitio web ASP.NET* y, antes de hacer clic sobre el botón *Aceptar*, estableceremos los parámetros siguientes:

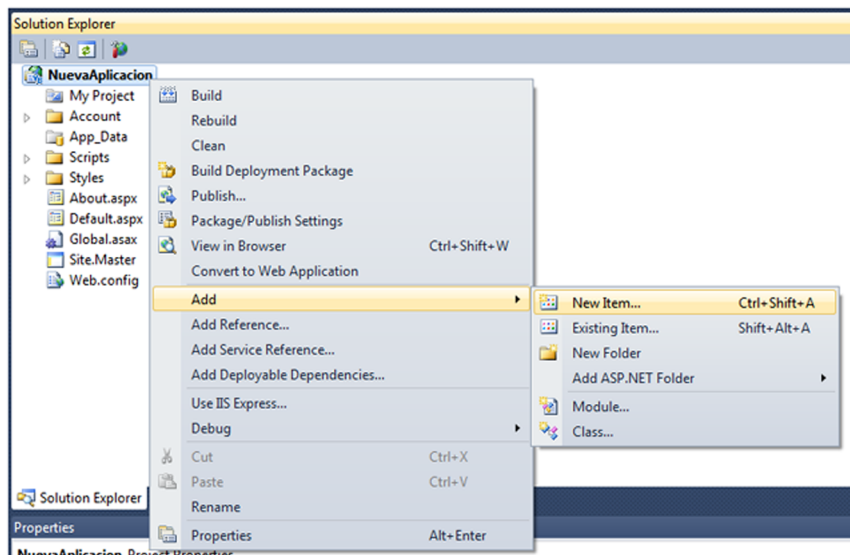
- La ubicación donde se alojará el proyecto, que podrá elegirse entre estas tres posibilidades: *Sistema de archivos*, *HTTP* y *FTP*. En nuestro caso siempre elegiremos *Sistema de archivos* e indicaremos la ruta donde se alojará el proyecto.
- El lenguaje de programación que se utilizará para codificar la lógica: *Visual Basic* o *C#*. En nuestro caso, *C#*.

Una vez establecida la configuración de la aplicación, un clic en el botón *Aceptar* generará el proyecto con los módulos iniciales.

3.1.3. Explorador de soluciones

Una vez generado el proyecto, los elementos que lo componen aparecerán en el Solution Explorer (figura 13), una herramienta fundamental para la administración de los proyectos.

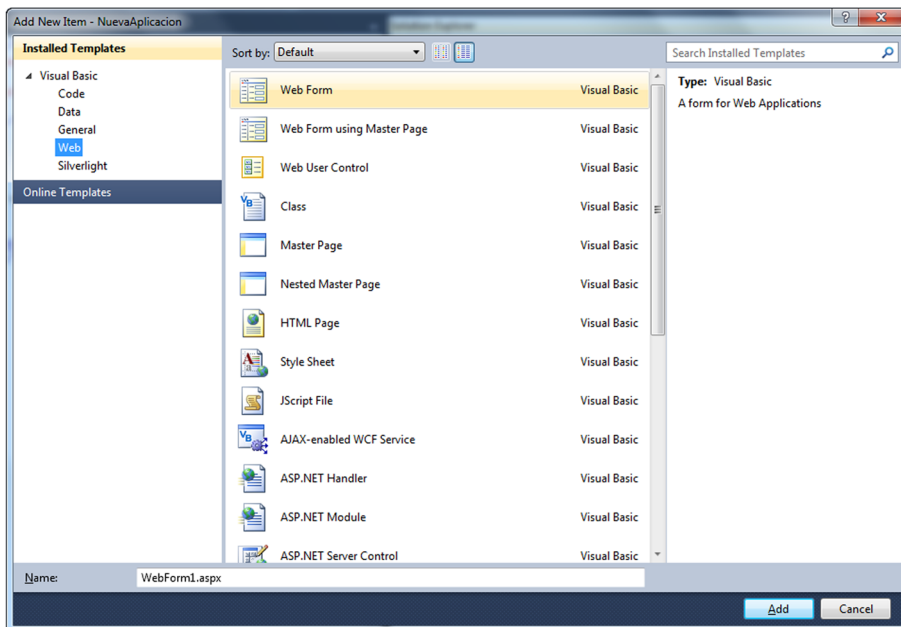
Figura 13. El Explorador de soluciones y el menú contextual asociado al proyecto



En principio, en esta ventana encontraremos tres elementos: la carpeta *App_Data*, la página *Default.aspx* y el módulo de configuración *web.config*. La página tiene asociado un archivo de código llamado *Default.aspx.cs*. Al seleccionar cualquiera de los elementos en la ventana *Properties*, aparecerán todas sus propiedades. Con el botón secundario del ratón podremos acceder al menú contextual, específico según el tipo de elemento que haya bajo el puntero del ratón. Un simple doble clic en un elemento de los mostrados en el Explorador de soluciones nos permitirá abrirlo en su editor/diseñador predeterminado. Recurriendo al menú contextual se puede optar por abrir el módulo en otro programa, eliminarlo del proyecto, copiarlo o llevar a cabo tareas específicas.

El menú contextual asociado al proyecto permite generarlo y ejecutarlo, así como agregar referencias y otros elementos. La opción *Add > New element* da paso a un cuadro de diálogo similar al de la figura 14, en el que se encuentran todos los tipos de elementos que es posible utilizar en una aplicación ASP.NET.

Figura 14. Cuadro de diálogo para agregar nuevos elementos a un proyecto

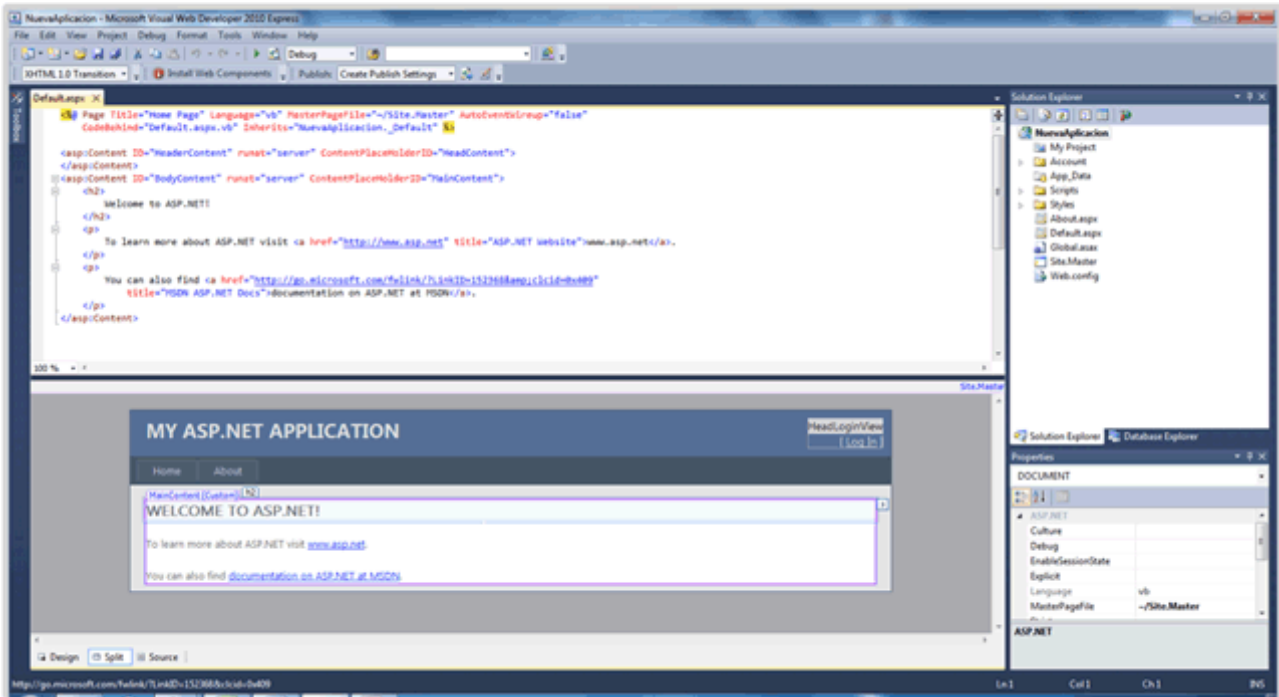


3.1.4. El diseñador de páginas

En cuanto creamos un nuevo proyecto, o hagamos doble clic sobre un módulo `.aspx` en la *Solution Explorer*, veremos cómo se abre el diseñador de páginas ASP.NET. Este se compone de un diseñador visual, similar a cualquier diseñador web, y de un editor de código XHTML/CSS. En la parte inferior las opciones *Diseño*, *Dividir* y *Código* permiten alternar entre tres vistas diferentes: únicamente los elementos de diseño, el diseñador y el editor, o únicamente el editor de código. En la figura 15 puede verse la segunda de las opciones, con la parte central dividida en dos secciones. La parte superior es el editor y la inferior el diseñador.

Mientras se trabaja en el diseñador (la parte inferior en la figura 15), es posible utilizar elementos habituales en el diseño de páginas web. La barra de botones que hay en la parte superior facilita la selección de tipo y tamaño de letra, colores de tinta y fondo, alineación, incluso de las listas numeradas y sin numerar, etc. También puede recurrirse a ventanas auxiliares, como *Aplicar estilos*, *Propiedades CSS* y *Administrar estilos*, que facilitan toda la gestión de los atributos que establecen el estilo de los elementos contenidos en la página. Los menús *Formato* y *Tabla*, activos solamente cuando está activo el diseñador, ofrecen opciones adicionales de aplicación de formato e inserción de tablas HTML.

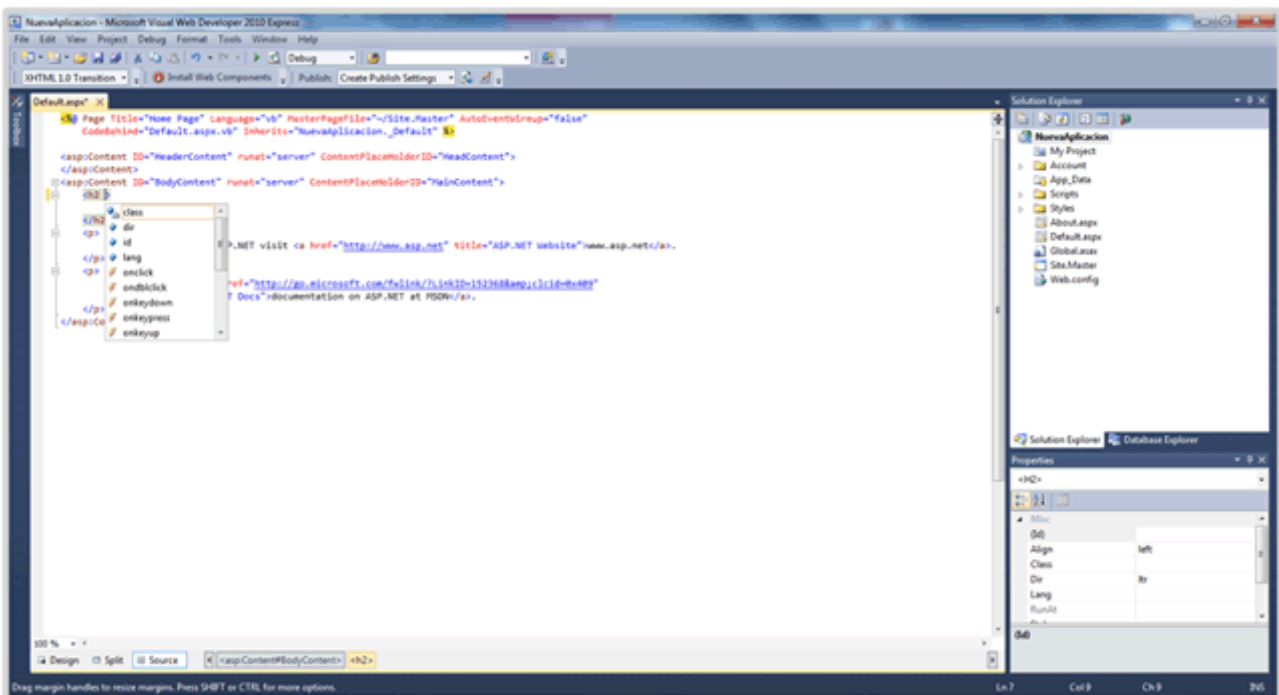
Figura 15. Diseñador de páginas y editor de código combinados en una misma vista



En cualquier momento es posible cambiar del diseñador al editor de código y trabajar sobre este. Los cambios llevados a cabo en el diseñador se reflejan inmediatamente en el código y viceversa. Al escribir código, en cuanto introduzcamos el carácter < para iniciar una etiqueta⁴, el editor nos ofrecerá ayuda inmediata a través de listas de elementos XHTML, atributos CSS, valores que permite cada atributo, etc. En la figura 16, por ejemplo, puede verse la lista de valores que permite el atributo style de una etiqueta h2.

⁽⁴⁾En inglés, tag.

Figura 16. No es necesario conocer de memoria los elementos y atributos de XHTML/CSS, el editor nos los recuerda



Aunque usando el editor XHTML/CSS es posible introducir cualquier elemento en una página, por regla general resultará mucho más cómodo hacerlo recurriendo a la ventana Toolbox (figura 17). Normalmente, la encontraremos oculta en el margen izquierdo del entorno.

En el Toolbox, los componentes aparecen agrupados en varias categorías: *Validation*, *AJAX Extensions*, *HTML*, etc. Los del grupo HTML no son más que elementos estándar de XHTML, en ocasiones con algunas propiedades preestablecidas. El resto son componentes de ASP.NET, es decir, porciones de código que se ejecutarán en el servidor, realizarán un cierto trabajo y generarán como resultado etiquetas XHTML/CSS.

La inclusión de cualquier tipo de componente en la página es una operación realmente sencilla: no hay más que tomarlo de la ventana Toolbox y arrastrarlo hasta la superficie del diseñador. En ese momento en el editor de código se introducirá la etiqueta correspondiente, generalmente del tipo `<asp:NombreComponente atributo="valor"...>`, y en el diseñador se apreciará el aspecto del control.

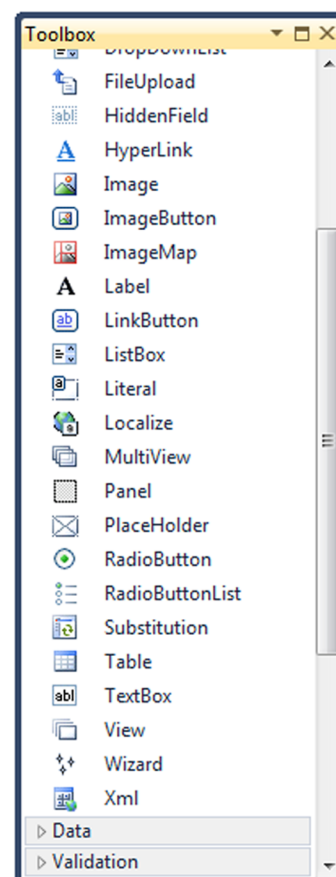


Figura 17. Cuadro de herramientas

Cada vez que se arrastra un componente desde el Toolbox hasta la superficie de diseño, lo que se obtiene es una copia, un objeto (una instancia de una clase), con unos atributos generales. Estos pueden personalizarse modificando las propiedades que exponen, utilizando para ello la ventana *Properties* (figura 18).

La modificación de ciertas propiedades, como las que afectan al estilo del componente, puede tener un reflejo inmediato en el aspecto del objeto en el diseñador. En otros casos, sin embargo, las propiedades tienen efecto únicamente durante la ejecución de la aplicación, como por ejemplo, la propiedad `AutoPostBack`, con la que se rige el comportamiento del componente una vez que está en el navegador del cliente, determinando si el cambio de una propiedad del propio componente provoca o no una comunicación con el servidor.

3.1.5. El editor de código C#

A medida que se va diseñando la interfaz de la aplicación, colocando componentes en el diseñador y personalizando sus propiedades, lo habitual es que también se vaya codificando la lógica, estableciendo el código que ha de ejecutarse cuando se envíe un formulario de datos, se cambie la selección de una lista, etc. La introducción del código en el servidor, que en su mayor parte estará asociado a eventos generados por la página o los componentes contenidos en ella, se lleva a cabo mediante un editor específico para C#.

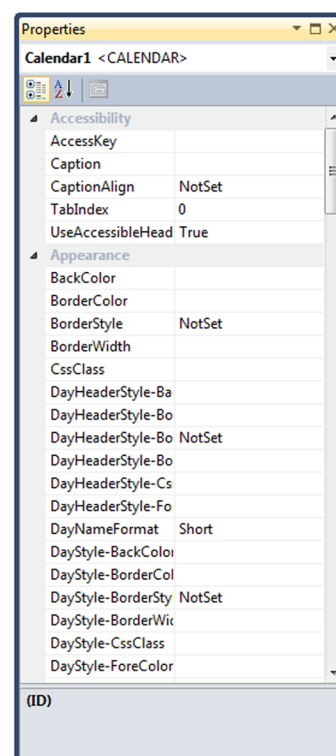


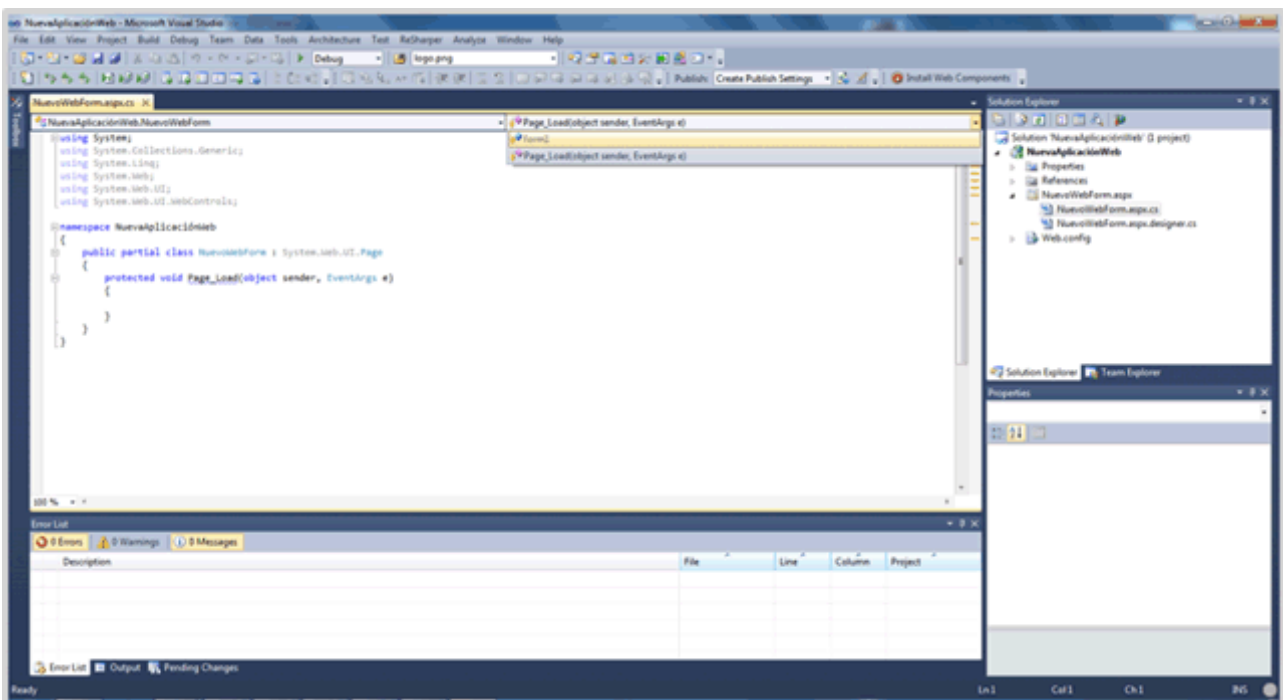
Figura 18. Listado de propiedades

Se puede abrir el editor de código de distintas formas:

- Haciendo doble clic sobre un componente para asociar código a su evento por defecto.
- Usando el botón *View Code* del *Solution Explorer* teniendo elegida cualquier página ASP.NET.
- Haciendo clic en el botón *Events* de la ventana *Properties* desde el diseñador, para acceder a la lista de eventos del componente que se tenga seleccionado, y luego eligiendo el evento al que se quiere asociar código.

El editor de código, como puede apreciarse en la figura 19, cuenta con dos listas desplegables en la parte superior. La de la izquierda permite seleccionar cualquiera de los elementos existentes en la página, momento en el que la lista de la derecha enumerará los eventos que se aplican a ese objeto. La selección de un evento provocará que el editor genere la cabecera del método encargado de responder al suceso, de forma que no hay más que introducir las sentencias que se quieren ejecutar en dicho caso.

Figura 19. El editor de código de C#



Al igual que el editor XHTML/CSS, este ofrece también una serie de ayudas enfocadas a facilitar el trabajo del programador y que se traducen en ventanas flotantes con información acerca de los miembros del objeto cuyo nombre se ha introducido, la lista de parámetros que acepta el método al que se pretende invocar, valores que pueden asignarse a una propiedad, etc.

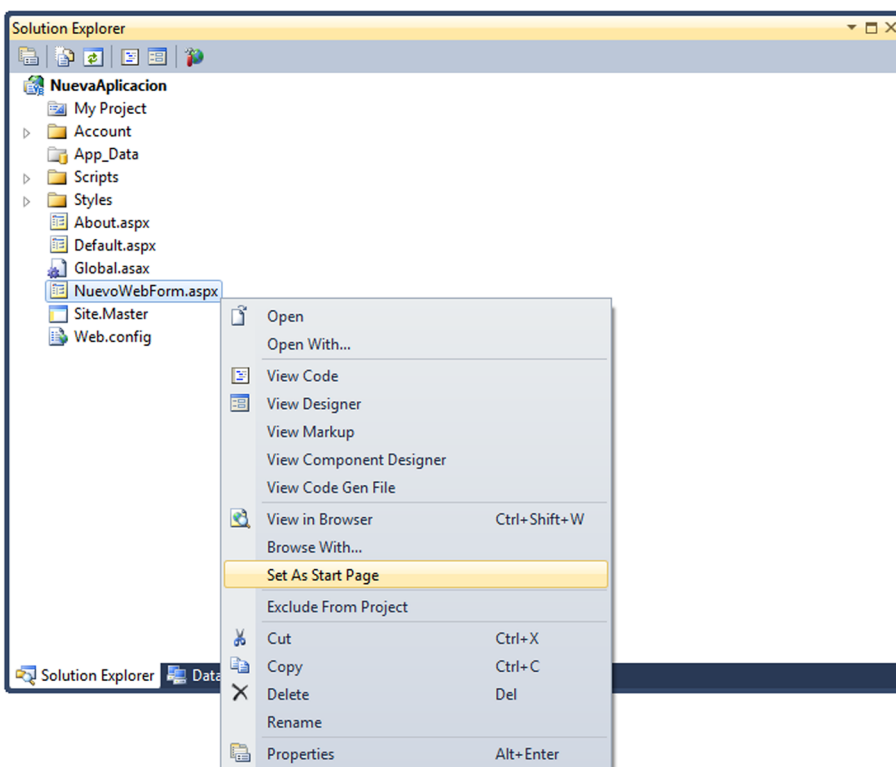
A diferencia de lo que ocurre con el código XHTML/CSS, el código C# no será interpretado ni ejecutado durante la fase de diseño de la aplicación, por lo que no tendrá ningún efecto sobre la interfaz de usuario que se construya paralelamente en el diseñador. Se trata de código que se procesará en el momento en que la aplicación sea ejecutada, de ahí que se almacene en un módulo independiente.

3.1.6. Ejecución y depuración de un sitio web

Dos de las tareas básicas de cualquier desarrollador web es la ejecución y depuración del sitio en el que trabaja. Con Visual Studio, ambas tareas se realizan muy fácilmente, lo cual es muy útil para el programador.

Para ejecutar un sitio web primero debemos especificar la página donde se iniciará la ejecución, ya que podemos tener varias páginas y en Visual Studio están todas al mismo nivel. Para especificar el punto de entrada de la aplicación web pulsamos el botón derecho sobre la página en cuestión y seleccionamos la opción *Set As Start Page*.


Figura 20. Establecer como página principal



Ahora, cuando ejecutemos el sitio web automáticamente, Visual Studio cargará la página seleccionada. Para ejecutar el sitio web sin depuración, lo podemos hacer de diferentes formas:

- Seleccionando la opción del menú *Debug > Start Without Debugging*.
- Pulsando **Ctrl + F5**.

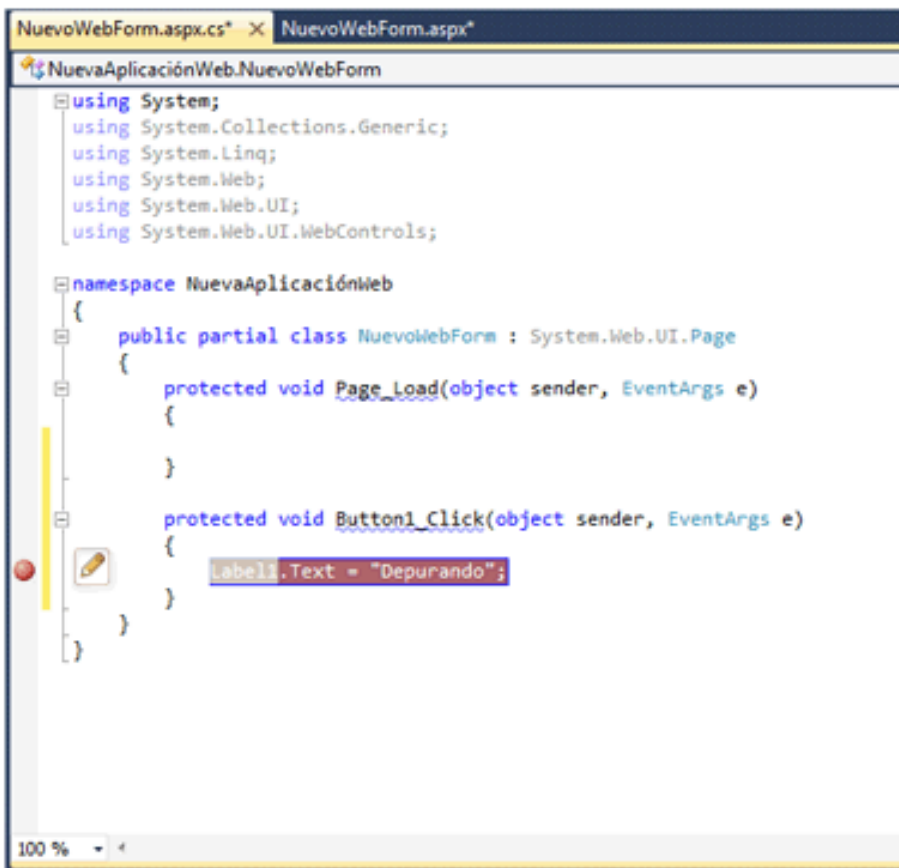
De esta forma podremos ver nuestra aplicación en ejecución, pero no podremos depurarla. Para ejecutar nuestro sitio web con depuración disponemos de varias opciones:

- Pulsando el botón  de la barra de menú superior.
- Seleccionando la opción del menú *Debug > Start Debugging*.
- Pulsando F5.

Visual Studio dispone de multitud de herramientas para la depuración de nuestro código. Las más importantes y útiles para nuestro curso son los *breakpoints* y los *watch* de variables.

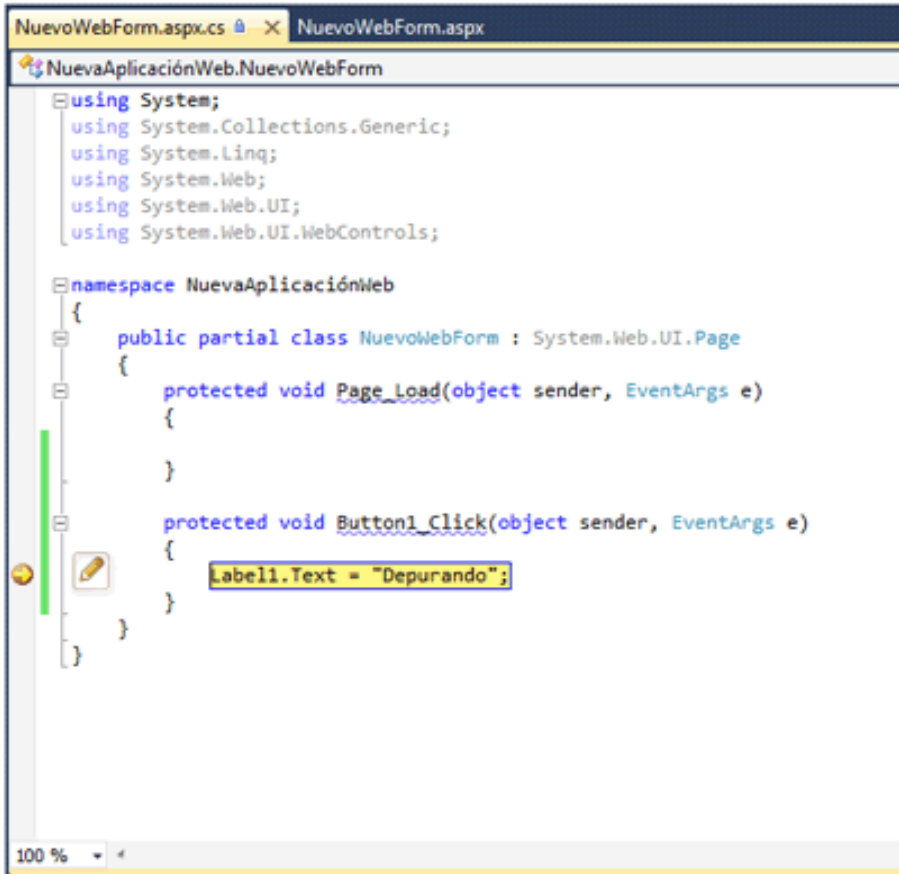
Un *breakpoint*, como ya sabemos, es la colocación de una interrupción en el flujo de ejecución. Para colocar un *breakpoint* en nuestro código, únicamente deberemos ir al archivo `.cs` y pulsar con el botón izquierdo del ratón sobre la columna izquierda de la fila deseada (figura 21).

Figura 21. *Breakpoint* en el archivo de código `.cs`

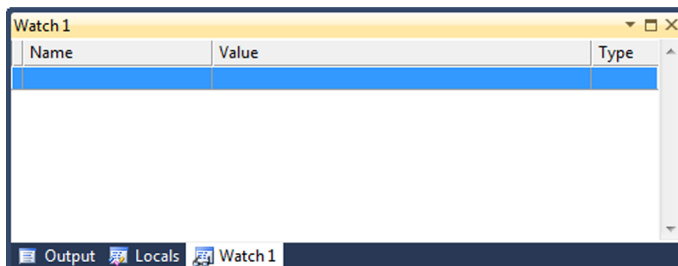


Si lanzamos la ejecución con depuración de nuestro sitio y clicamos sobre el botón, podremos ver cómo el flujo se para en el *breakpoint* que hemos insertado (figura 22).

Figura 22. Depurando la aplicación

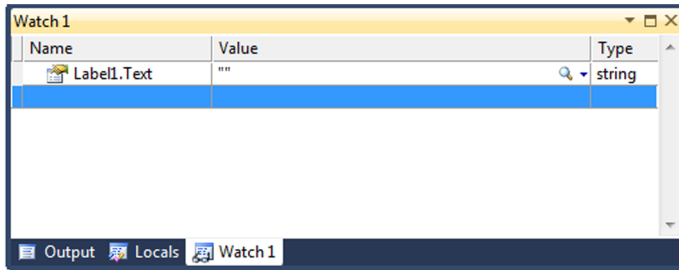


Una vez el *breakpoint* ha saltado, podemos ver el valor de todas las variables que actualmente tenemos en nuestro contexto. El valor de las variables lo podemos consultar en el apartado inferior *Watch* (figura 23).

Figura 23. Ventana *Watch* de variables

Si pulsamos doble clic sobre la columna nombre podemos insertar el nombre de la variable que queremos consultar y automáticamente en la columna valor se actualizará el contenido de dicha variable (figura 24).

Figura 24. Inspeccionando el valor de una variable





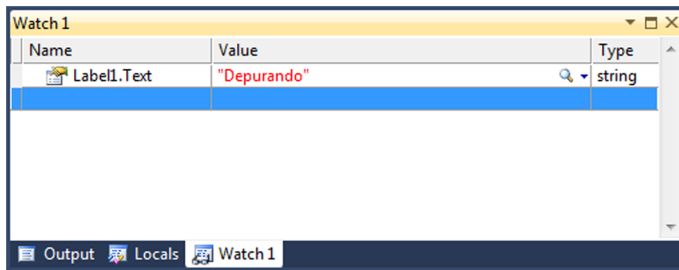

Para avanzar el flujo de ejecución una vez parado en nuestro *breakpoint*, disponemos de los botones   de la barra de menú superior. Si avanzamos una línea el flujo de ejecución veremos cómo la variable cambia de valor automáticamente (figura 25).

Figura 25. Inspeccionando el valor de una variable



Finalmente, si queremos retomar la ejecución normal de la aplicación, únicamente debemos pulsar otra vez el botón .

3.1.7. Base de datos compacta con Visual Studio

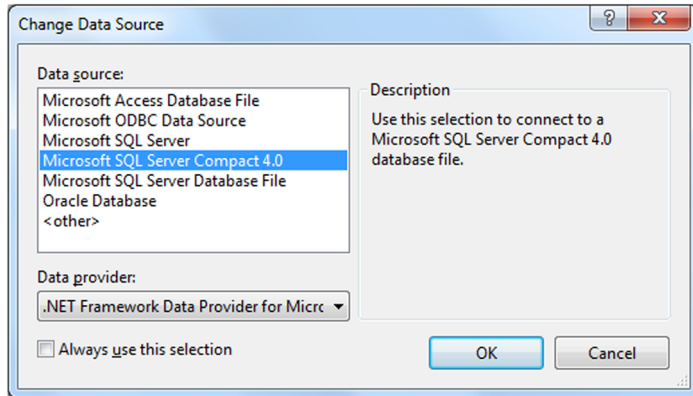
En la mayoría de aplicaciones web es necesaria una base de datos de donde recoger la información. Para aplicaciones extensas puede ser necesaria una base de datos convencional que necesite de un servidor de base de datos para su funcionamiento, pero para proyectos pequeños como son los nuestros, una base de datos compacta (de un solo archivo sin necesidad de servidor) es más que suficiente. Visual Studio nos permite la creación de ambos tipos de base de datos desde el mismo entorno de programación.

Creación de la base de datos

Nosotros nos centraremos en la creación de una base de datos compacta mediante Visual Studio.

Primero de todo accederemos al apartado del menú *Tools > Connect to Database*. Seleccionamos *Microsoft SQL Server Compact 4.0* y pulsamos *OK* (figura 26).

Figura 26. Tipo de base de datos



En la siguiente ventana, asignamos un nombre a nuestra base de datos en el apartado *Database*, por ejemplo “NuevaBaseDatos”. En el apartado *Password* introducimos una contraseña, por ejemplo “uoc”, y pulsamos *Create* (figura 27). Una vez creada la base de datos, pulsamos *OK* y veremos cómo en el Database Explorer nos aparece una nueva base de datos *NuevaBaseDatos.sdf* (figura 28).

Figura 27. Añadiendo la conexión a la base de datos

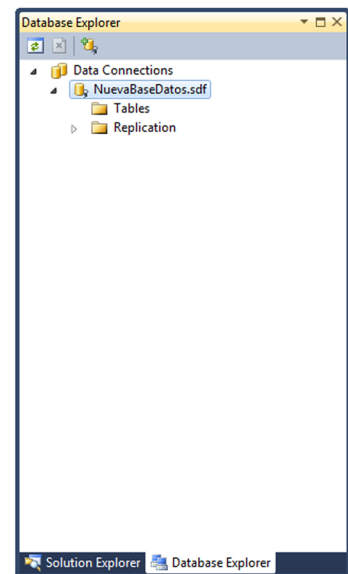
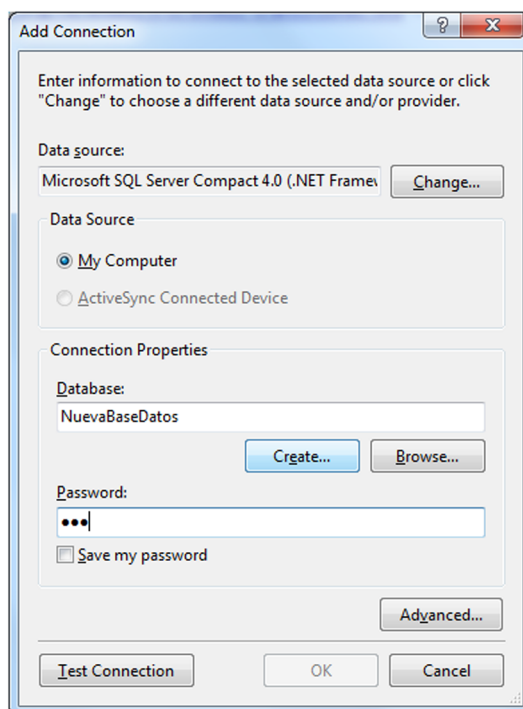
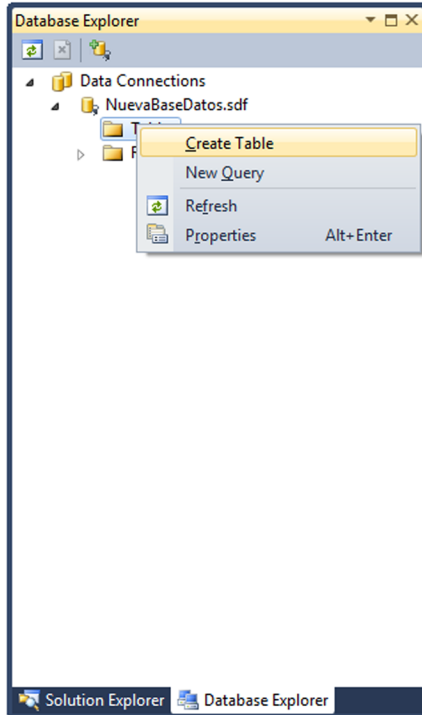


Figura 28. Base de datos compacta

Creación de tablas

Para la creación de tablas, únicamente deberemos pulsar el botón derecho sobre *Tables* y seleccionar *Create Table* (figura 29).

Figura 29. Creación de tablas



A continuación nos aparece una ventana donde podremos especificar el nombre y los campos de la tabla. Vemos que es muy parecido a otros gestores de bases de datos; para cada columna debemos especificar el nombre del campo, el tipo, si debe ser único, si puede estar vacío y si es clave primaria. Una vez rellenados los campos, pulsamos OK (figura 30). Esto nos generará nuestra primera tabla (figura 31).

Figura 30. Formulario de nueva tabla

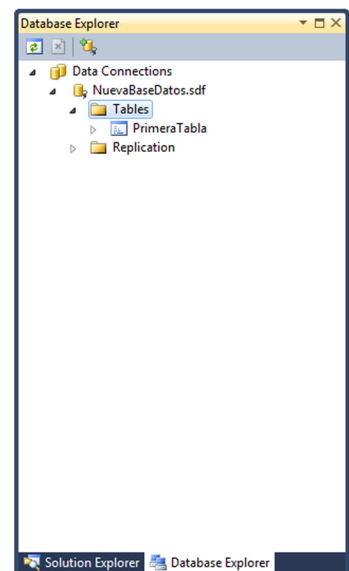
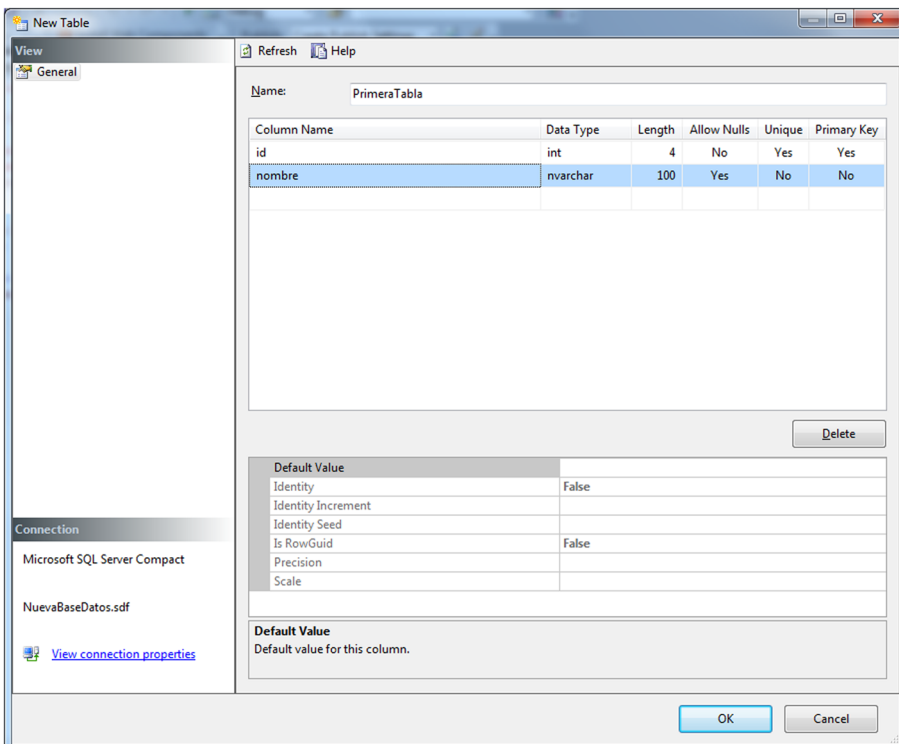


Figura 31. Nueva tabla

Nos queda poder crear claves foráneas entre nuestras tablas. Para ello creamos una nueva tabla llamada “SegundaTabla” (figura 32). Pulsamos el botón derecho sobre dicha tabla una vez creada y seleccionamos la opción *Table Properties* (figura 33).

Figura 32. Edición de tabla

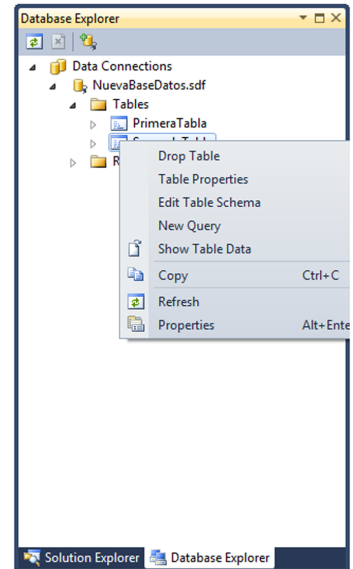
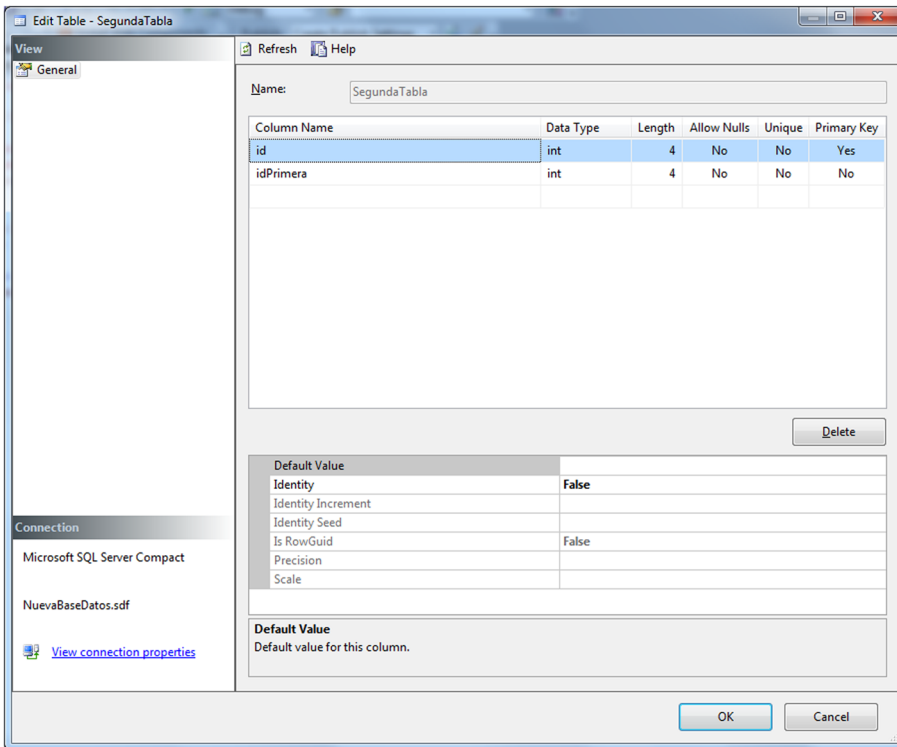
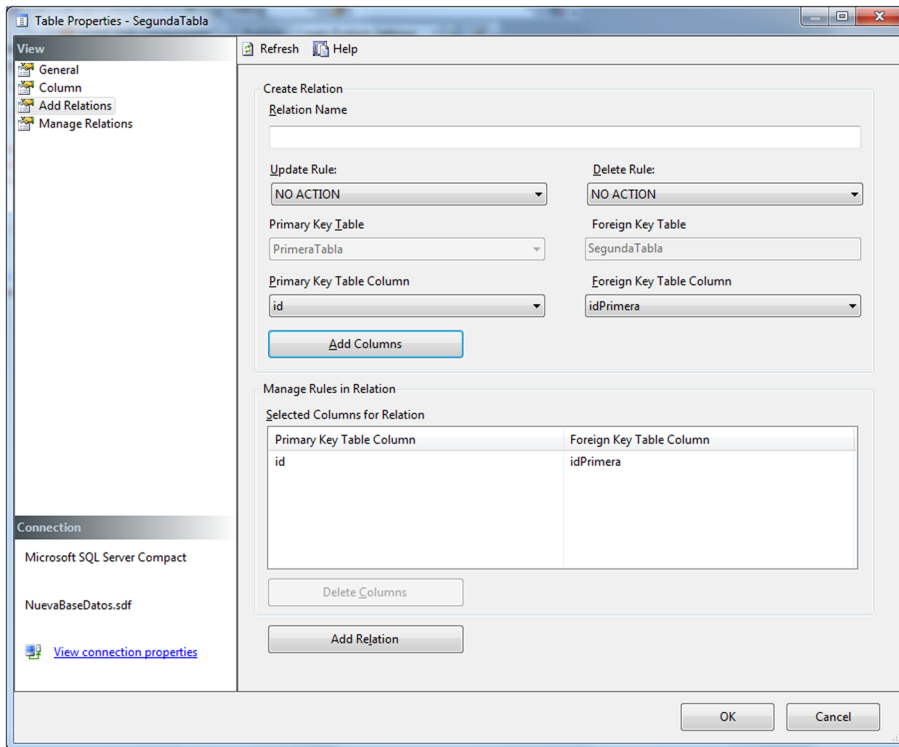


Figura 33. Edición de tabla

Dentro de las propiedades vamos al apartado *Añadir relaciones* y rellenamos los campos de dicho apartado teniendo en cuenta que la columna izquierda corresponde a la clave primaria de la tabla origen y la columna derecha corresponde a la tabla actual, donde debemos escoger la columna que será clave foránea. Pulsamos *Add Columns* y automáticamente se añadirá la nueva relación de clave foránea entre las tablas (figura 34).

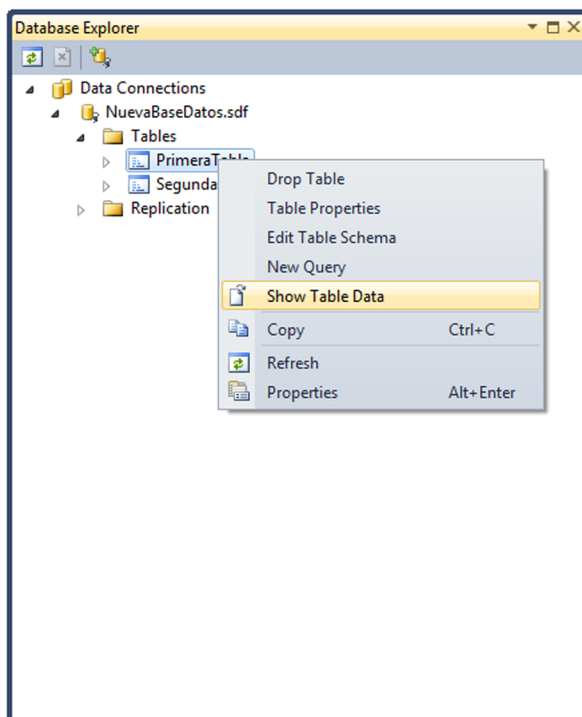
Figura 34. Añadiendo clave foránea



Inserción y edición de datos

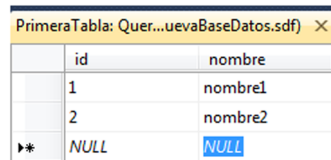
Una vez creada la base de datos con sus tablas relacionadas, únicamente nos queda insertar los datos necesarios. Para ello pulsamos el botón derecho sobre la tabla donde queremos insertar datos y seleccionamos la opción *Show Data Table* (figura 35).

Figura 35. Añadiendo datos a la tabla



Finalmente veremos una tabla que podremos editar añadiendo o modificando filas a nuestro antojo (figura 36).

Figura 36. Añadiendo datos a la tabla



	id	nombre
	1	nombre1
	2	nombre2
▶*	NULL	NULL

Manejo de la base de datos desde código

1) Selección de datos

Para manejar la base de datos desde el código C#, ya sea para insertar, editar o consultar, lo primero que debemos hacer es abrir una conexión con la misma. A continuación veremos un pequeño ejemplo donde seleccionaremos datos de la base de datos y los cargaremos en un combo.

Primero de todo definimos en el archivo `aspx` el combo donde se cargarán los datos y un botón para lanzar el evento que consultará la base de datos.

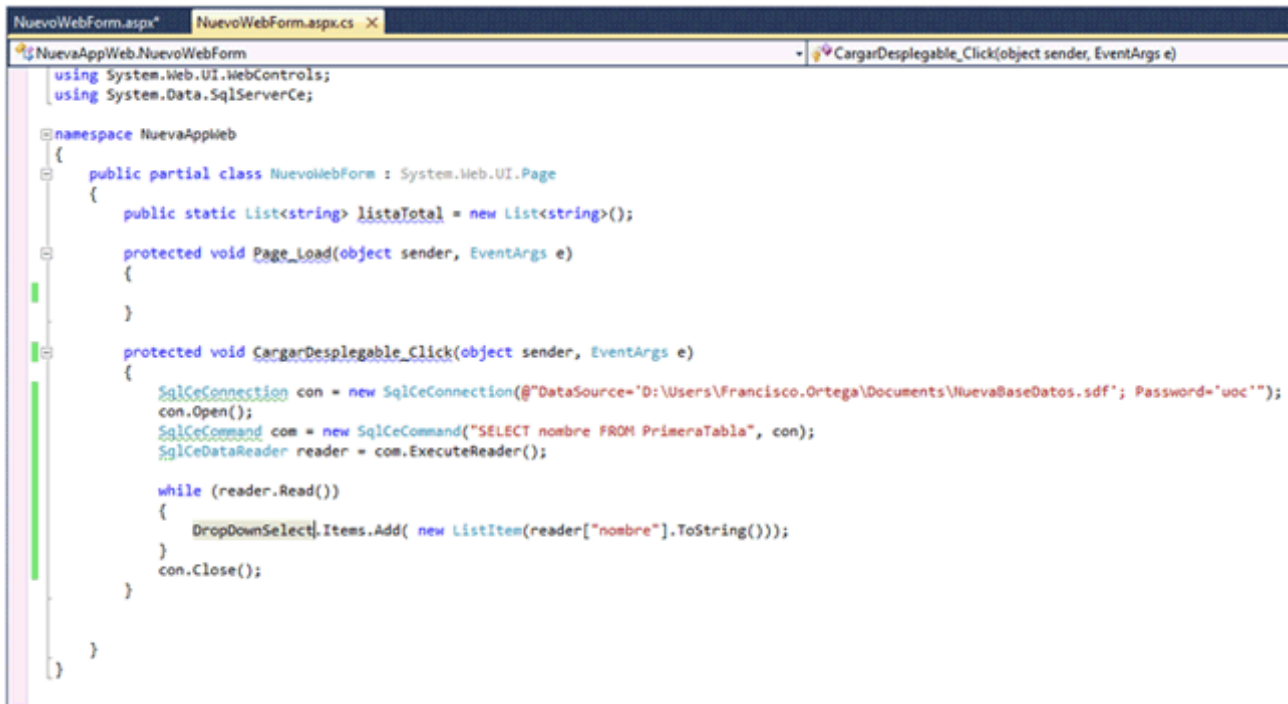
Figura 37



```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs" Inherits="NuevaAppWeb.NuevoWebForm" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Button runat="server" onclick="CargarDesplegable_Click" Text="Cargar Desplegable"/>
<asp:DropDownList runat="server" ID="DropDownSelect"></asp:DropDownList>
</div>
</form>
</body>
</html>
```

A continuación, en el archivo `aspx.cs` definimos el evento del botón con el código de conexión y selección de base de datos. Vemos cómo se define la biblioteca de Sql Server Compact con la instrucción `using System.Data.SqlServerCe`. Según la versión de Visual Studio, es posible que no tengáis la biblioteca (*dll*) a la que se hace referencia. La encontraréis en el espacio de ficheros del aula.

Figura 38



```
using System.Web.UI.WebControls;
using System.Data.SqlServerCe;

namespace NuevaAppWeb
{
    public partial class NuevoWebForm : System.Web.UI.Page
    {
        public static List<string> listaTotal = new List<string>();

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

        protected void CargarDesplegable_Click(object sender, EventArgs e)
        {
            SqlConnection con = new SqlConnection(@"DataSource='D:\Users\Francisco.Ortega\Documents\NuevaBaseDatos.sdf'; Password='uoc'");
            con.Open();
            SqlCommand com = new SqlCommand("SELECT nombre FROM PrimeraTabla", con);
            SqlDataReader reader = com.ExecuteReader();

            while (reader.Read())
            {
                DropDownList.Items.Add( new ListItem(reader["nombre"].ToString()));
            }
            con.Close();
        }
    }
}
```

Comentaremos línea por línea el código que podemos ver en el evento de botón del servidor:

```
SqlConnection con = new SqlConnection(@"DataSource='D:\Users\Francisco.Ortega\
Documents\NuevaBaseDatos.sdf'; Password='uoc'");
```

Primero de todo conectamos con la base de datos con la clase `SqlConnection` pasándole la ruta del archivo de base de datos creado anteriormente y la contraseña de la misma.

```
con.Open();
```

Es necesario abrir la conexión con la base de datos para poder empezar a realizar operaciones con ella.

```
SqlCommand com = new SqlCommand("SELECT nombre FROM PrimeraTabla", con);
```

Creamos la consulta que realizaremos a la base de datos con el objeto `SqlCommand` pasándole la conexión que hemos creado.

```
SqlDataReader reader = com.ExecuteReader();
```

Para ejecutar la consulta utilizamos la función `ExecuteReader()`. Esta función nos devuelve los resultados de la consulta que almacenamos en un objeto de tipo `SqlDataReader`.

```
while (reader.Read())
```



```
{  
    DropDownList.Items.Add( new ListItem(reader["nombre"].ToString()));  
}
```

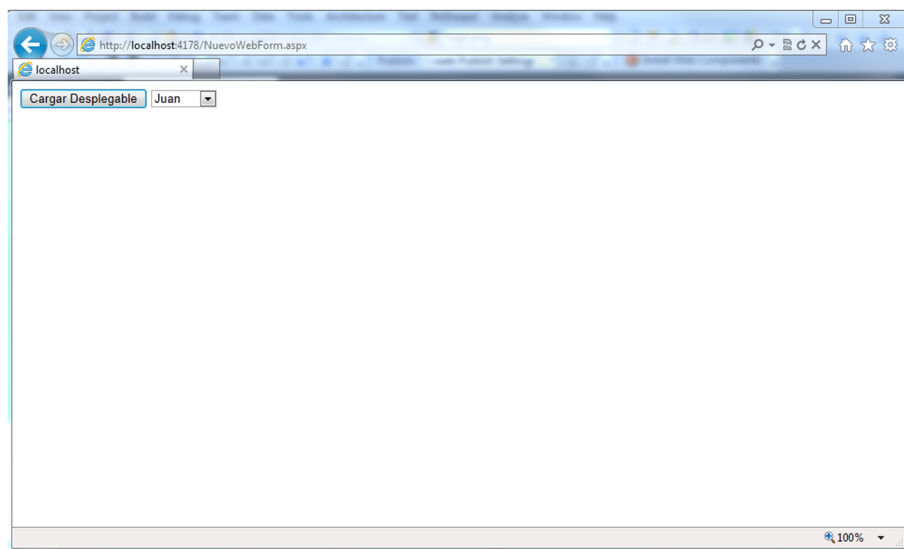
Vamos recorriendo el `SqlCeDataReader` con un `while` extrayendo en cada iteración una fila del resultado con la función `Read()`. Una vez extraída la fila accederemos a los campos de bases de datos mediante la notación `reader["nombreCampo"]`.

```
con.Close();
```

Una vez realizadas todas las operaciones con la base de datos, cerramos la conexión para no sobrecargarla.

Si ejecutamos la aplicación, podemos ver cómo el combo se carga con la consulta de la base de datos al pulsar el botón *Cargar Desplegable* (figura 39).

Figura 39



2) Edición de datos

Para la edición de datos de la base de datos, igual que en la selección, debemos abrir una conexión con la misma.

Continuando con el ejemplo anterior, añadiremos un `TextBox`, donde insertaremos el valor que queremos que reemplace a la selección del combo.

Figura 40



```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs" Inherits="NuevaAppWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Button runat="server" onclick="CargarDesplegable_Click" Text="Cargar Desplegable"/>
<asp:DropDownList runat="server" ID="DropDownSelect"></asp:DropDownList>
|
<asp:TextBox ID="TextBoxEditor" runat="server"></asp:TextBox>
<asp:Button runat="server" onclick="EditarSeleccion_Click" Text="Editar selección"/>
</div>
</form>
</body>
</html>

```

Además, como podemos ver, hemos añadido un botón que lanzará el evento de edición del campo del combo con el valor del TextBox.

Figura 41



```

protected void EditarSeleccion_Click(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection(@"DataSource='D:\Users\Francisco.Ortega\Documents\NuevaBaseDatos.sdf';
    Password='uoc'");
    con.Open();
    var itemSelected = DropDownSelect.SelectedItem;

    if(!String.IsNullOrEmpty(TextBoxEditor.Text))
    {
        SqlCommand com = new SqlCommand("UPDATE PrimeraTabla SET nombre='"+
            TextBoxEditor.Text + "' WHERE nombre='"+ itemSelected.Text + "'", con);
        com.ExecuteNonQuery();
    }

    con.Close();
}

```

Podemos ver que la parte de conexión y desconexión con base de datos es idéntica a la del ejemplo anterior, únicamente varía la creación y ejecución de la consulta.

```

SqlCommand com = new SqlCommand("UPDATE PrimeraTabla SET nombre='" +
    TextBoxEditor.Text + "' WHERE nombre='" + itemSelected.Text + "'", con);

```

Utilizando el mismo objeto SqlCommand creamos query de Update pasándole la conexión a la base de datos.

```

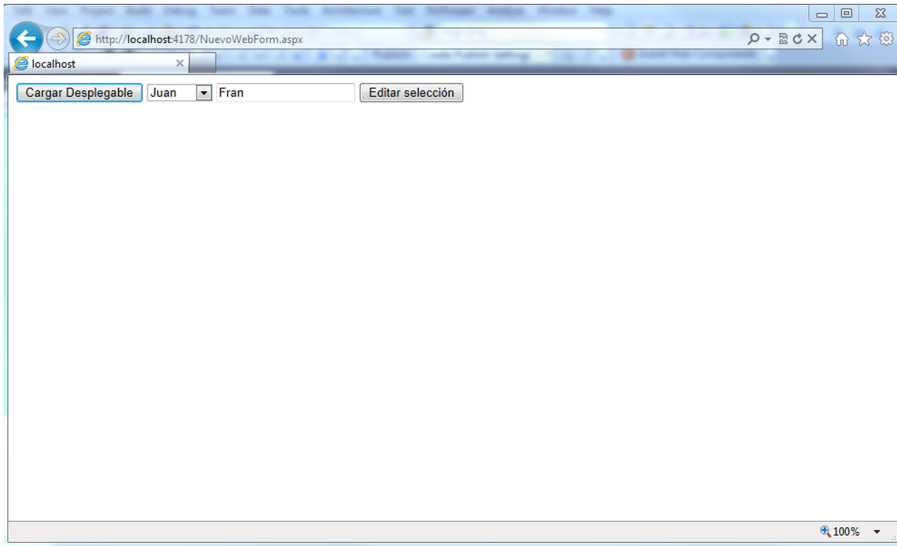
com.ExecuteNonQuery();

```

Para una edición de datos se debe utilizar la función `ExecuteNonQuery()`, en lugar de `ExecuteReader()`.

Si ejecutamos la aplicación, veremos cómo, si insertamos un valor en el `TextBox`, seleccionamos un valor en el `select` y pulsamos *Editar selección*, el valor del desplegable se actualiza cuando volvemos a pulsar *Cargar Desplegable*.

Figura 42



3) Inserción de datos

La inserción de datos es prácticamente idéntica a la edición. Continuando con el ejemplo, añadiremos un `TextBox`, donde especificaremos la cadena de caracteres que insertaremos en la base de datos.

Figura 43

```
NuevoWebForm.aspx x NuevoWebForm.aspx.cs
Client Objects & Events (No Events)
<? Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs" Inherits="NuevaAppWeb.NuevoWebForm" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Button runat="server" onclick="CargarDesplegable_Click" Text="Cargar Desplegable"/>
<asp:DropDownList runat="server" ID="DropDownSelect"></asp:DropDownList>

<asp:TextBox ID="TextBoxEditor" runat="server"></asp:TextBox>
<asp:Button runat="server" onclick="EditarSeleccion_Click" Text="Editar selección"/>

<asp:TextBox ID="TextBoxInsertar" runat="server"></asp:TextBox>
<asp:Button runat="server" onclick="InsertarNombre_Click" Text="Insertar nombre"/>
</div>
</form>
</body>
</html>
```

Además, añadimos un botón que lanzará el evento de inserción en la base de datos.

Figura 44

```
protected void InsertarNombre_Click(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection(@"DataSource='D:\Users\Francisco.Ortega\Documents\NuevaBaseDatos.sdf';
    Password='uoc'");
    con.Open();

    if (!String.IsNullOrEmpty(TextBoxInsertar.Text))
    {
        SqlCommand com = new SqlCommand("INSERT INTO PrimeraTabla (nombre)VALUES(@nombre)", con);
        com.Parameters.AddWithValue("@nombre", TextBoxInsertar.Text);
        com.ExecuteNonQuery();
    }

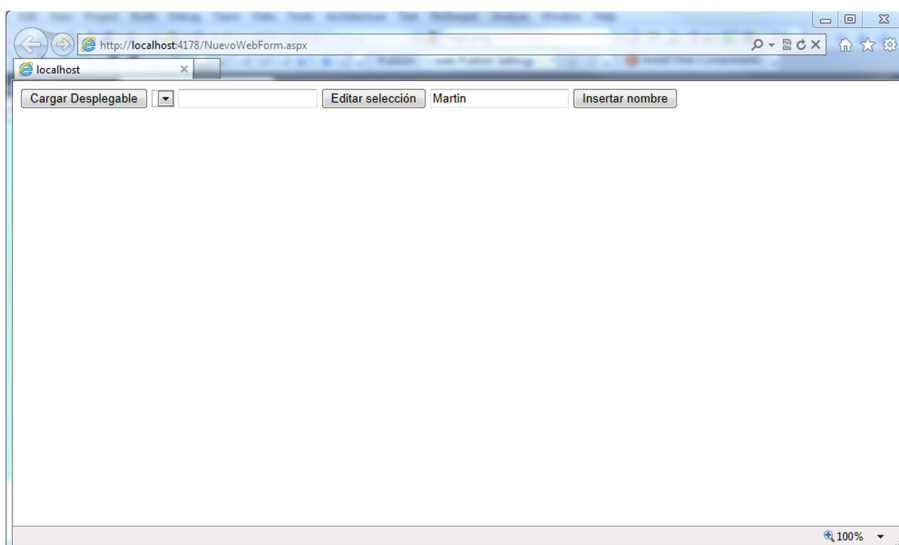
    con.Close();
}
```

El código que varía con respecto a la edición de datos es el que concierne a la creación de la query de inserción.

```
SqlCommand com = new SqlCommand("INSERT INTO PrimeraTabla (nombre)VALUES(@nombre)", con);
com.Parameters.AddWithValue("@nombre", TextBoxInsertar.Text);
```

Primero creamos la query de inserción especificando los valores que insertaremos con la notación @nombreCampo. A continuación especificamos dichos valores con la función Parameters.AddWithValue() pasándole el identificado @nombreCampo y el valor que queremos insertar. Si ejecutamos la aplicación, insertamos un valor en el TextBox y pulsamos el botón *Insertar nombre*, veremos cómo la cadena se inserta en la base de datos. Para poder verlo, refrescaremos el combo pulsando *Cargar Desplegable*.

Figura 45



4. Introducción a ASP.NET

ASP.NET es una de las tecnologías que integra .NET mediante la cual los desarrolladores pueden crear aplicaciones web.

En este apartado se explicarán las características más destacables de ASP.NET, se hará una breve introducción histórica desde los principios de ASP.NET hasta ASP.NET 4.0. A continuación, se comenzará a ver una introducción a la tecnología ASP.NET, explicando lo que son los WebForm, los controles de servidor o las MasterPage.

4.1. Características principales de ASP.NET

En ASP.NET hay siete factores o características clave que lo diferencian de los demás productos de Microsoft y de las plataformas de la competencia. Estas características son las siguientes:

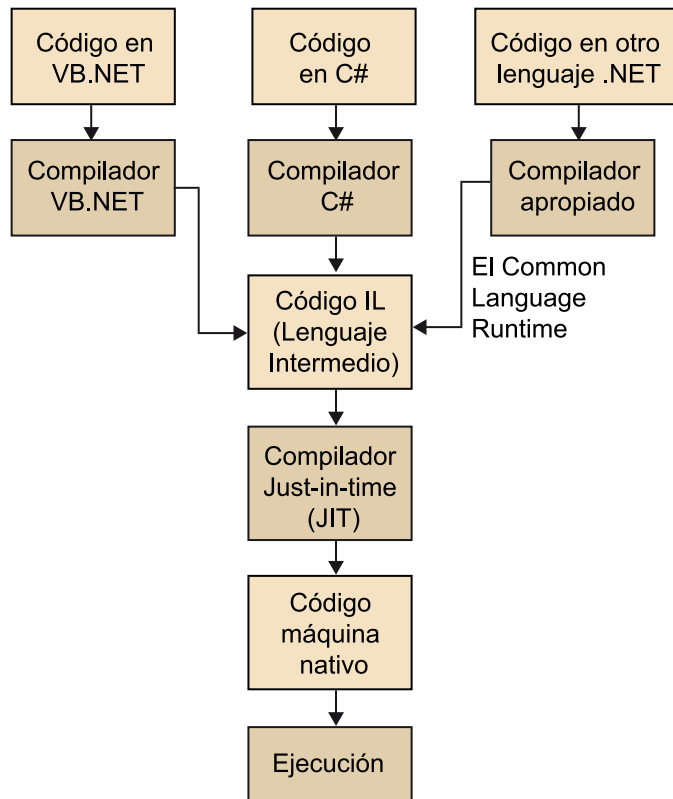
1) ASP.NET está integrado con el entorno de desarrollo .NET Framework.

El entorno .NET Framework es un conjunto de miles de clases, estructuras, interfaces, etc., que está organizado según su funcionalidad. El modo de utilizar las clases de .NET en ASP.NET es el mismo que en cualquier otro tipo de plataforma .NET. En otras palabras, la utilización del entorno de desarrollo .NET es la misma en una aplicación web que en una de escritorio. Esta característica es verdaderamente importante porque implica que Microsoft ofrece las mismas herramientas a los desarrolladores web que a los programadores de aplicaciones cliente.

2) ASP.NET es compilado, no interpretado.

Las aplicaciones ASP.NET, al igual que todas las aplicaciones .NET, siempre se compilan. Esta compilación pasa por dos etapas bien diferenciadas. En la primera etapa, el código C# se transforma en un lenguaje intermedio llamado Microsoft Intermediate Language (MSIL). Esta etapa permite que las plataformas .NET sean multilinguaje, es decir, que se pueda programar en diferentes lenguajes de programación, como C# o Visual Basic. La segunda etapa de compilación transforma el código MSIL en código máquina. En la figura 46 podemos ver un diagrama con las diferentes etapas de compilación.

Figura 46. Etapas de compilación de ASP.NET



3) **ASP.NET es multilinguaje.** Como hemos visto, gracias a la primera etapa, cualquier plataforma .NET puede ser multilinguaje. Únicamente necesitamos un compilador que transforme el lenguaje en el cual programamos al lenguaje intermedio MSIL. Los dos lenguajes de programación por excelencia en .NET son C# y Visual Basic.

4) **ASP.NET es orientado a objetos.** En ASP.NET se pueden explotar todas las características de una programación orientada a objetos. Por ejemplo, se pueden crear clases reutilizables, estandarizar el código utilizando interfaces o extender clases mediante la herencia.

5) **ASP.NET es compatible con todos los navegadores.** Uno de los mayores desafíos a los que se enfrenta un desarrollador web es la gran cantidad de navegadores que hay en el mercado. ASP.NET aborda este problema de forma bastante innovadora: anima a los desarrolladores a utilizar una serie de controles de servidor. Estos controles son funcionales en cliente solo si este soporta todas sus características.

6) **ASP.NET es fácil de desplegar y configurar.** Uno de los mayores quebraderos de cabeza a los que se tiene que enfrentar un desarrollador es el despliegue de una aplicación web en un servidor real.

7) Otro aspecto destacable de ASP.NET es su **fácil configuración**, ya que toda la plataforma está unificada en un solo archivo `web.config`. En este archivo podemos configurar desde permisos de acceso a cadenas de conexión de base de datos.

4.2. La evolución de ASP.NET

Cuando Microsoft lanzó al mercado ASP.NET rápidamente se convirtió en el estándar para desarrollo web con tecnología .NET y en un duro competidor para las otras plataformas de desarrollo web.

Desde entonces, Microsoft ha creado varias versiones de ASP.NET. En los siguientes subapartados se explica cómo ASP.NET ha ido evolucionando a lo largo de los años.

4.2.1. ASP.NET 1.0 y 1.1

La idea central de ASP.NET fue crear un modelo de páginas web llamado *formularios web*. Básicamente, cuando un navegador solicita una página ASP.NET, crea en el servidor un objeto `Page` y tantos objetos como controles tenga dicha página. De esta forma, el desarrollador puede modificar aspectos de la página en tiempo de ejecución.

4.2.2. ASP.NET 2.0

La versión 2.0 mantuvo el mismo modelo de formularios web y se concentró en la adición de nuevas características de alto nivel como:

- Páginas maestras. Las `MasterPage`, o páginas maestras⁵, son plantillas para la reutilización de código, como por ejemplo en encabezados o pies de página.
- Navegación. Se incluyen diferentes controles de navegación como el árbol, el mapa del sitio o el hilo de Ariadna.
- Seguridad. Incluye soporte para el almacenamiento de credenciales de usuario y nuevos controles de servidor para el acceso⁶, el registro de usuario o el recordatorio de contraseña.
- Origen de datos. La versión 2.0 permite enlazar controles de servidor con diferentes fuentes de datos como bases de datos o XML.

⁽⁵⁾En inglés, *master pages*.

⁽⁶⁾En inglés, *login*.

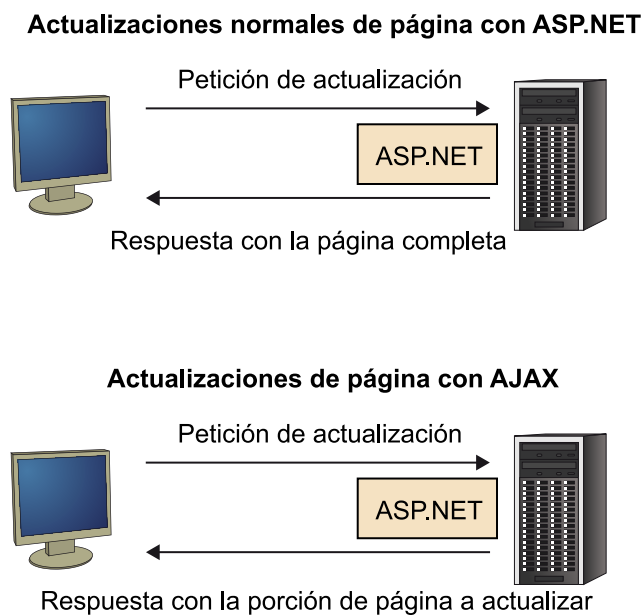
4.2.3. ASP.NET 3.5

La principal novedad que se puede destacar de la versión 3.5 es la introducción de dos nuevas tecnologías: Linq y Ajax.

La primera tecnología es un conjunto de ampliaciones para C# o Visual Basic, que permite manipular datos en memoria (por ejemplo, una lista) de la misma manera que lo haríamos con consultas a una base de datos.

En las versiones anteriores de ASP.NET, para cada petición desde el navegador, el servidor debía procesar toda la página entera y volverla a enviar al servidor. Este proceso resultaba lento e ineficiente, además de ser poco interactivo desde el punto de vista del usuario. Para solucionar este problema se implementó la tecnología AJAX para ASP.NET, que mediante JavaScript es capaz de realizar recargas parciales de una página web. La figura 47 ilustra las diferencias entre las peticiones tradicionales y las peticiones AJAX.

Figura 47. Peticiones tradicionales y peticiones Ajax



4.2.4. ASP.NET 4.0

En la versión 4.0 de ASP.NET, las nuevas características aportadas son algo más sutiles. Algunas de ellas son:

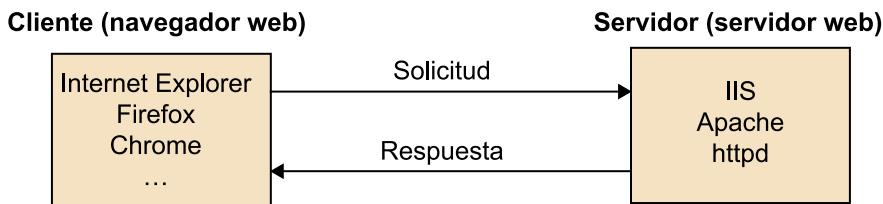
- Renderización en XHTML. Después de procesar la página correspondiente a la petición, se envía al cliente en el estándar XHTML.
- Control de ViewState. Se incorpora la activación/desactivación del ViewState. El ViewState es un objeto que ASP.NET va transmitiendo entre las llamadas asíncronas donde se almacena información de la página.

- Control de gráficos. Se incorpora un control para la realización de diferentes gráficos, como gráficos de líneas, barras, curva, superficie, circulares, de anillos y gráficos de punto.
- Enrutamiento. Tecnología para poder redirigir peticiones web al igual que hacen la mayoría de los patrones de diseño modelo-vista-controlador.

4.3. Estructura de una aplicación ASP.NET

La estructura de una aplicación web clásica consiste en una arquitectura cliente-servidor según la cual existe un servidor y uno o varios clientes que acceden al mismo (figura 48).

Figura 48. Arquitectura cliente servidor



Dicha estructura es utilizada también en una aplicación ASP.NET. Además, las aplicaciones ASP.NET son ejecutadas en el servidor por una capa de software, conocida habitualmente como *motor de ASP.NET*, que es la encargada de interpretar la solicitud que le facilita el servidor web y generar un objeto `HttpRequest` con toda la información. En este objeto estarán contenidos, por ejemplo, todos los datos introducidos en los formularios por el usuario, el navegador utilizado o la información de estado de la aplicación.

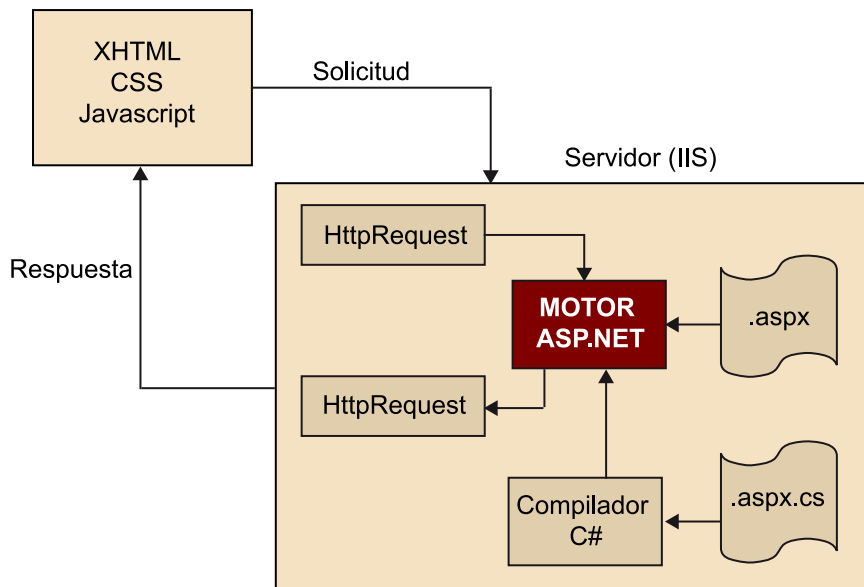
Cada página de un proyecto ASP.NET se compone al menos de dos módulos: uno con extensión `.aspx`, en el que está contenido el código XHTML estático, y otro con la extensión `.aspx.cs`, que aloja el código C# correspondiente a la lógica de la aplicación.

La primera vez que se recibe una solicitud para una página, el motor de ASP.NET la compila y obtiene una versión lista para ejecutarse, que se aloja en memoria. En las solicitudes siguientes la respuesta será inmediata, ya que se dispone de una versión compilada.

En la figura 49 se muestra un esquema de bloques de los elementos que forman una aplicación ASP.NET. Como se puede ver, el cliente siempre obtendrá como respuesta una combinación de XHTML, CSS y JavaScript, lenguajes válidos para cualquier navegador web con independencia de la plataforma que se utilice para realizar la petición.

Figura 49. Arquitectura ASP.NET

Cliente (navegador web)



4.3.1. Tipos de archivos en ASP.NET

Las aplicaciones web en ASP.NET pueden tener diferentes tipos de archivos, algunos de los cuales se muestran a continuación:

- **.aspx**: son las páginas web en ASP.NET. Contienen la interfaz de usuario. El punto de entrada para los usuarios siempre es uno de estos archivos.
- **.ascx**: son los controles de usuario en ASP.NET. Son similares a las páginas web, pero no son accesibles directamente desde el navegador, sino que siempre forman parte de una página. Se utilizan para reutilizar código y evitar así la repetición del mismo.
- **web.config**: archivo de configuración en XML. Desde aquí se configuran aspectos de seguridad, memoria, conexiones a bases de datos, etc.
- **global.asax**: es el fichero global de la aplicación. Aquí se pueden definir variables globales, accesibles desde cualquier página, y eventos propios de la aplicación.
- **.cs**: archivo de código C#. Nos permite separar la lógica de la aplicación de la interfaz de usuario.

Además de estos archivos, las aplicaciones ASP.NET pueden contener otros archivos comunes en muchas aplicaciones web, como archivos JavaScript, HTML, CSS o imágenes.

4.4. ¿Qué són los WebForm?

Las Pages o, como oficialmente se conocen, los WebForm son la parte más importante de una aplicación ASP.NET. Estos WebForm generan todo el código HTML de respuesta a cualquier petición que solicita un cliente desde el navegador. Cada WebForm que añadimos a nuestra aplicación será una página accesible por el usuario. Un WebForm está compuesto por dos archivos de código, un archivo .aspx, donde reside todo el código de la interfaz de usuario, y un archivo .aspx.cs, con todo el código relacionado al WebForm como eventos, validaciones, etc.

A continuación podemos ver un ejemplo de un WebForm donde se muestran ambos ficheros:

Figura 50. Ficheros .aspx y .cs de un WebForm



```
NuevoWebForm.aspx X
Client Objects & Events
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>

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

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

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

NuevoWebForm.aspx.cs X NuevoWebForm.aspx
NuevaAplicaciónWeb.NuevoWebForm
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

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

        }
    }
}
```

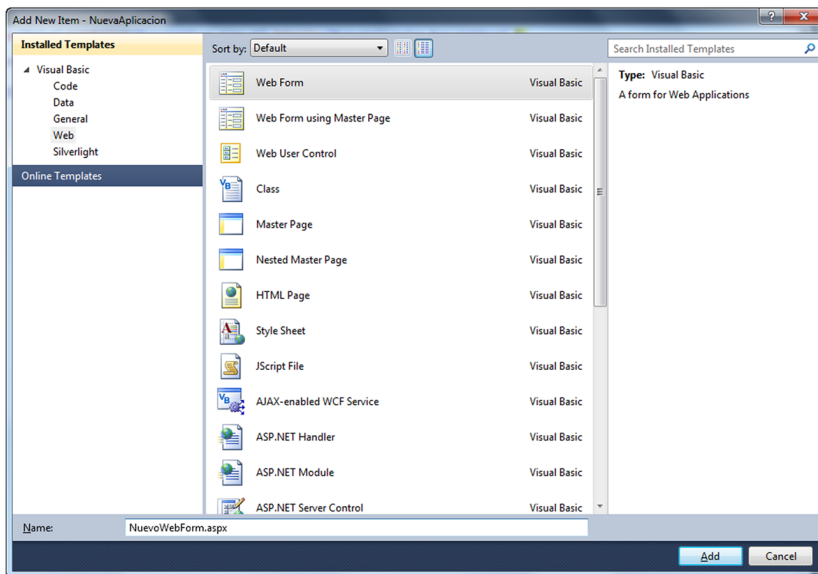
Internamente, un WebForm o Page se genera gracias a la clase `Page`, como hemos podido ver tanto en la cabecera del fichero `.aspx` como en la herencia del fichero `.cs`. Posteriormente se dedicará un apartado para explicar más profundamente dicha clase, una de las más importantes de la biblioteca de clases de .NET.

4.4.1. Añadiendo un WebForm a nuestro sitio web

Para añadir un WebForm a un sitio web creado previamente, únicamente debemos seguir los siguientes pasos:

- Pulsamos el botón derecho sobre la raíz del proyecto, o sobre cualquier carpeta donde queramos que se aloje el WebForm, para abrir el menú contextual y a continuación pulsamos *Add > New item*.
- A continuación seleccionamos la plantilla *WebForm*, le damos un nombre a nuestra página y pulsamos *Add* (figura 51).

Figura 51. Añadiendo un WebForm



- Si desplegamos en el *Solution Explorer* nuestro WebForm, podemos ver los archivos `.aspx` y `.aspx.cs` (figura 52).
- Finalmente, podemos explorar ambos archivos en el editor de código correspondiente pulsando doble clic en cada uno de ellos.

4.5. La clase Page

Cada página de una aplicación web en ASP.NET hereda de la clase `Page`. Mediante esta herencia nuestras páginas adquieren una serie de propiedades y métodos que podemos utilizar en los ficheros `.cs`.

A continuación enumeraremos algunas de las propiedades más importantes:

- **IsPostBack**: booleano que indica si es la primera vez que la página se ha cargado.
- **EnableViewState**: booleano. Nos permite decidir si queremos guardar la información de estado de los controles entre peticiones.
- **Application**: colección que contiene información compartida entre todos los usuarios del sitio web.
- **Session**: colección que contiene información de un usuario en particular que se mantiene entre diferentes peticiones.
- **Request**: colección que contiene información sobre la petición actual como información del navegador o los parámetros enviados.
- **Response**: objeto `HttpResponse` relacionado con la respuesta a una determinada petición. Se puede utilizar para redireccionar al usuario a otra página, enviar *cookies* al navegador, etc.

Un caso muy común donde se utiliza una propiedad de la clase `Page` es en la redirección del flujo hacia otra página web. Concretamente, se utiliza la propiedad `Response`.

Para ver un ejemplo insertaremos en un WebForm únicamente un `Button` con su respectivo evento de servidor.

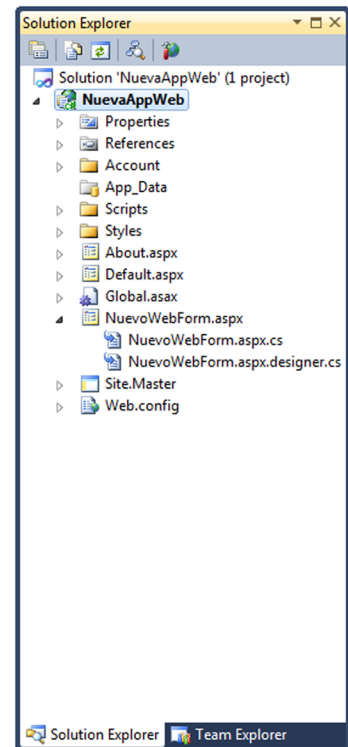


Figura 52. Añadiendo un WebForm

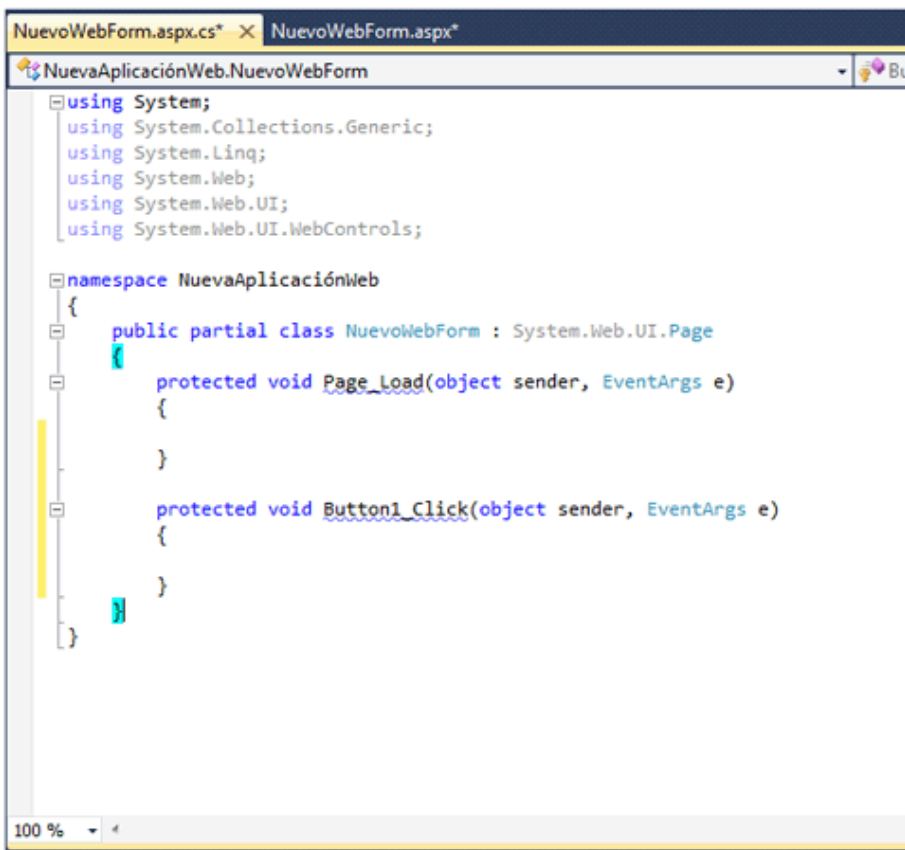
Figura 53. Código .aspx de un WebForm



```
Client Objects & Events (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Button ID="Button1" runat="server" Text="Button" />
</div>
</form>
</body>
</html>
```

Figura 54. Código .cs de un WebForm



```
NuevoWebForm.aspx.cs* X NuevoWebForm.aspx*
NuevaAplicaciónWeb.NuevoWebForm
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

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

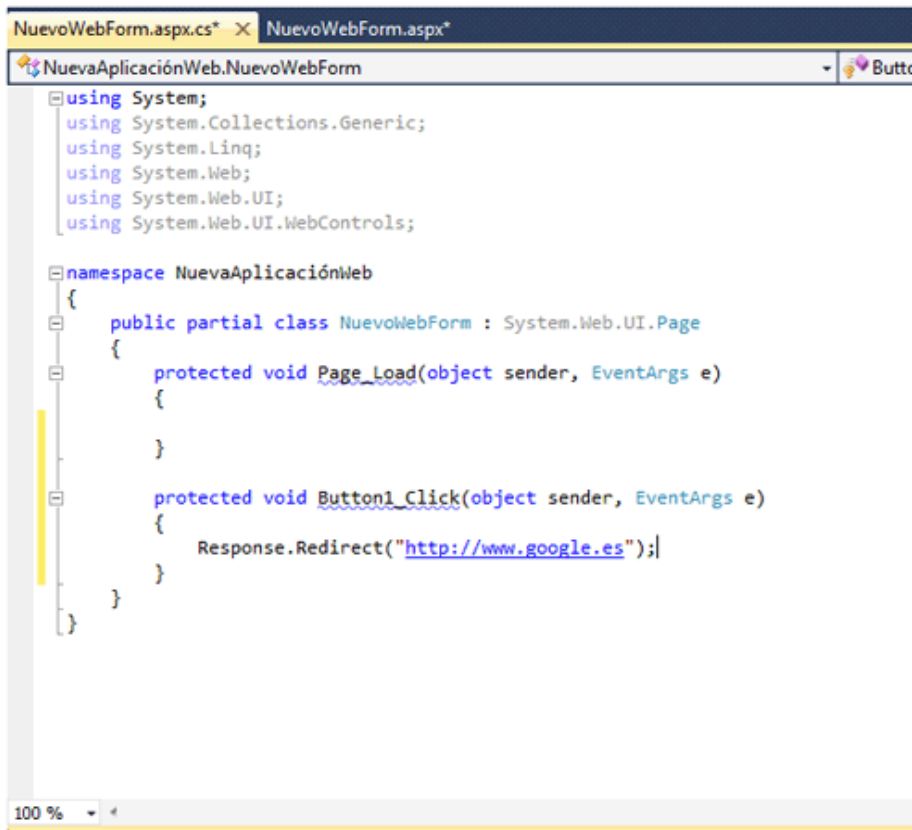
        }

        protected void Button1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

En el evento `click` del botón hacemos una redirección hacia otra página web de la siguiente manera:

Figura 55. Código .cs de un WebForm



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

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

        protected void Button1_Click(object sender, EventArgs e)
        {
            Response.Redirect("http://www.google.es");
        }
    }
}
```

Si ejecutamos, veremos que cuando pulsamos en el botón (figura 56) el flujo se redirecciona hacia la URL que hemos puesto en el `Response.Redirect` (figura 57).

Figura 56. Ejecución de un WebForm

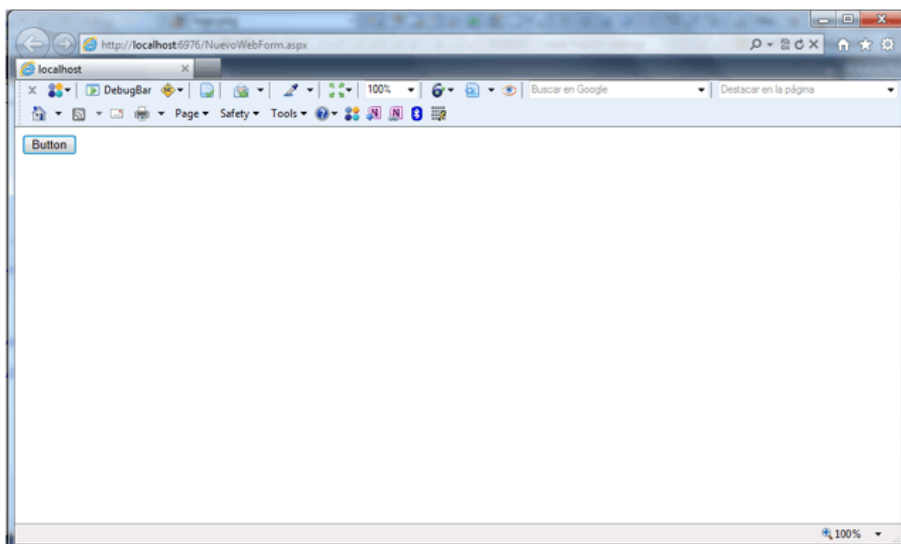
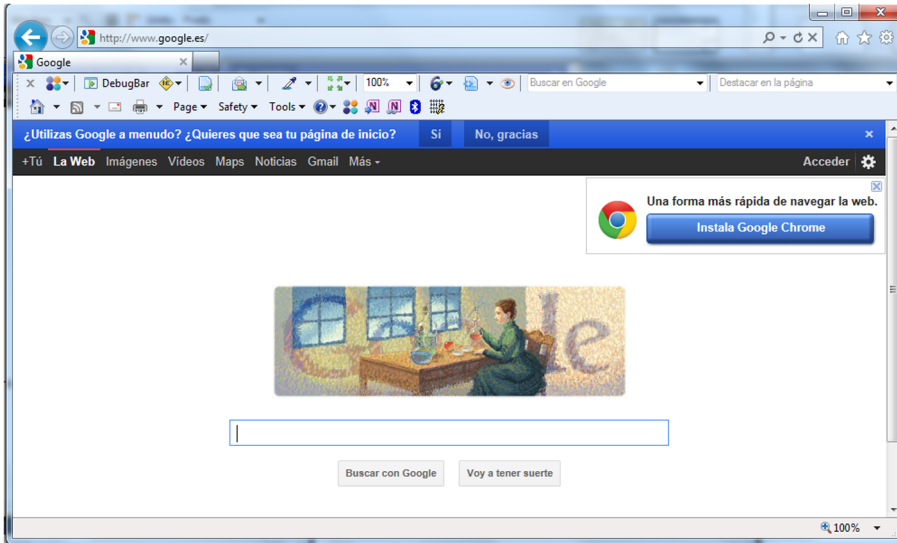


Figura 57. Ejecución de un WebForm



Este ejemplo nos puede servir para hacer redireccionamiento entre las páginas de nuestro sitio y así poder navegar entre ellas.

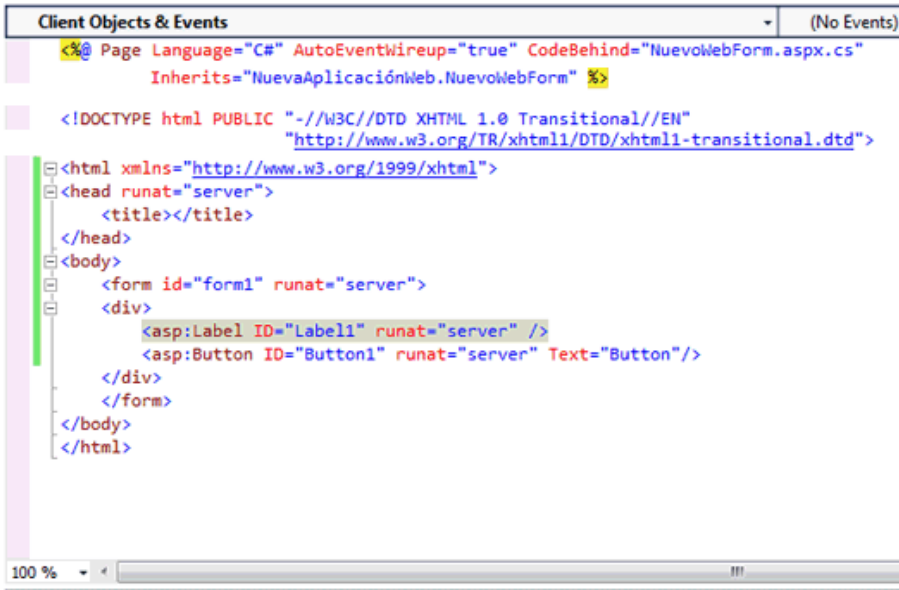
4.6. El ViewState

En una aplicación de escritorio de Windows, una parte de la memoria del ordenador es utilizada para almacenar el estado de la misma. En una aplicación web, el funcionamiento es totalmente diferente. Un cliente se conecta al servidor web, solicita una página y la conexión finaliza cuando la página se entrega, y se descartan todos los objetos que haya en memoria. Por esta razón, surge el problema del almacenamiento de cierta información del estado de la aplicación entre diferentes peticiones de un cliente.

Una de las maneras de almacenar esta información en ASP.NET es hacerlo en el ViewState. ASP.NET inserta automáticamente el ViewState como un campo oculto en cada renderización HTML de una página. De esta forma se conservan los datos entre diversas peticiones de servidor.

A continuación (figuras 58, 59 y 60) podemos ver un ejemplo de un contador que almacena las veces que un usuario pulsa un botón.

Figura 58. Código .cs de un ViewState

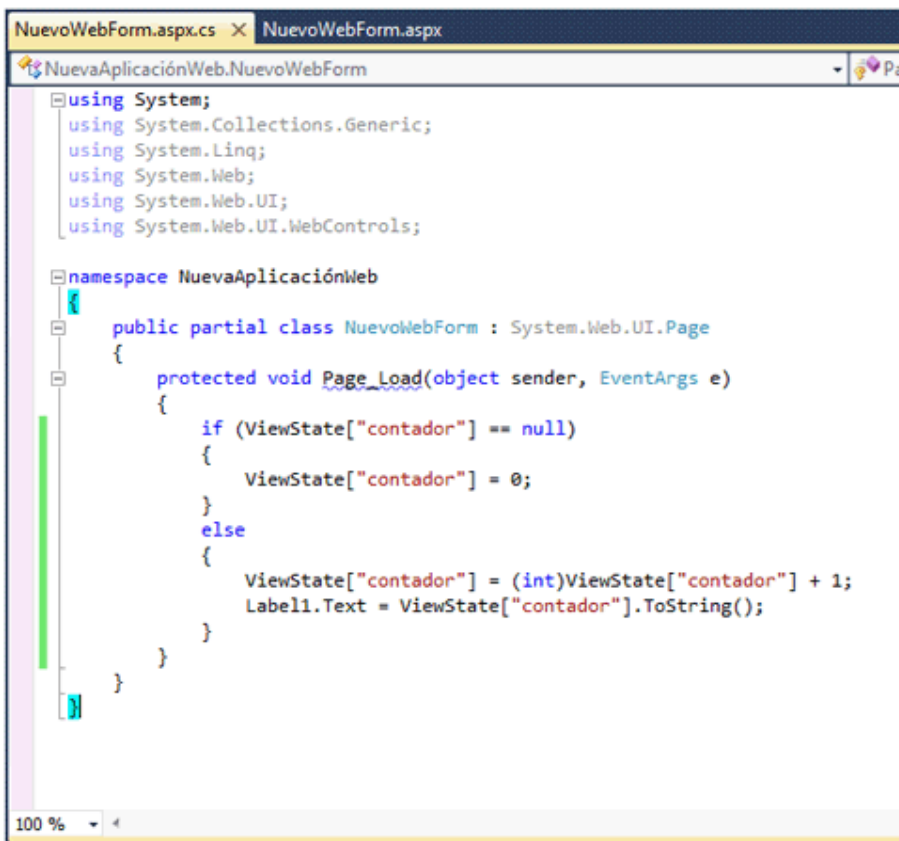


```
Client Objects & Events (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" />
        <asp:Button ID="Button1" runat="server" Text="Button"/>
    </div>
    </form>
</body>
</html>
```

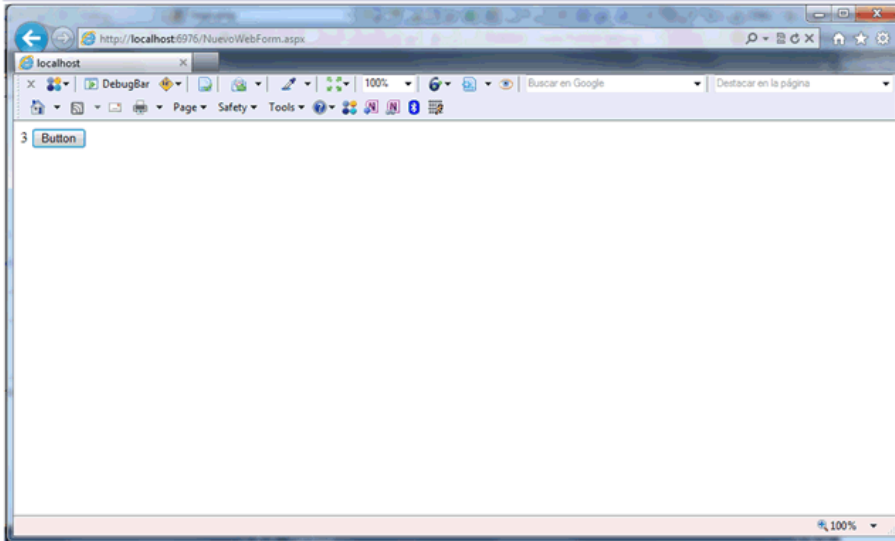
Figura 59. Código .cs de un ViewState



```
NuevoWebForm.aspx.cs X NuevoWebForm.aspx
NuevaAplicaciónWeb.NuevoWebForm
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace NuevaAplicaciónWeb
{
    public partial class NuevoWebForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (ViewState["contador"] == null)
            {
                ViewState["contador"] = 0;
            }
            else
            {
                ViewState["contador"] = (int)ViewState["contador"] + 1;
                Label1.Text = ViewState["contador"].ToString();
            }
        }
    }
}
```

Figura 60. Ejecución de un ViewState



4.7. Controles de servidor

ASP.NET introdujo hace varios años un modelo innovador para la creación de páginas web. Anteriormente, para el diseño de páginas web dinámicas se introducían etiquetas en el propio fichero HTML y, según la interpretación de estas etiquetas, se producía una salida de HTML u otro. Esta técnica, además de resultar tediosa, producía un código poco extensible y reaprovechable.

ASP.NET solventó este problema introduciendo el concepto de *controles de servidor*.

Los **controles de servidor** se crean y se configuran como objetos. Una vez configurados por el desarrollador, generan automáticamente su propio código HTML.

Algunas características de los controles de servidor son:

- Generan su propia interfaz de usuario. Se genera el código HTML en tiempo de ejecución y se envía al cliente.
- Retienen su estado. Guardan información de sí mismos entre diferentes peticiones.
- Disponen de eventos de servidor. Por ejemplo, un botón puede ejecutar una función de servidor al ser pulsado.

ASP.NET ofrece una gran cantidad de controles de servidor que se pueden clasificar en diferentes grupos. Todos ellos los encontraremos en el Toolbox de Visual Studio. Alguno de los grupos más importantes son:

- a) Controles de servidor HTML: grupo donde encontramos los elementos HTML estándares como los enlaces (<a>), los *inputs* (<input>), etc. Para convertir un elemento HTML estándar en un control de servidor, únicamente debemos indicar el atributo `runat=server`.
- b) Controles web: controles con las mismas funcionalidades que algunos elementos HTML, pero que disponen de propiedades y métodos que facilitan su declaración y acceso. Algunos ejemplos son el Hiperlink, ListBox, Button, etc.
- c) Controles Rich: controles que amplían las funcionalidades de los elementos HTML e incluso son capaces de generar código JavaScript propio. Un ejemplo es el TreeView.
- d) Controles de validación: controles que permiten aplicar normas o reglas a las entradas de datos de los usuarios.
- e) Controles de datos: controles diseñados para mostrar grandes cantidades de datos y para la edición, ordenación y paginación de los mismos. El ejemplo más claro es el control GridView.
- f) ASP.NET Ajax Controls: estos controles permiten al programador utilizar la tecnología AJAX sin necesidad de escribir código JavaScript en el lado del cliente.

4.7.1. Controles de servidor HTML

Como hemos dicho, los controles HTML son exactamente iguales que los elementos HTML comunes. Podemos encontrar todos estos controles en el Toolbox, dentro del apartado *HTML* (figura 61).

Si arrastramos cualquier control HTML y lo soltamos en el diseñador de páginas veremos cómo automáticamente se genera el código HTML correspondiente.

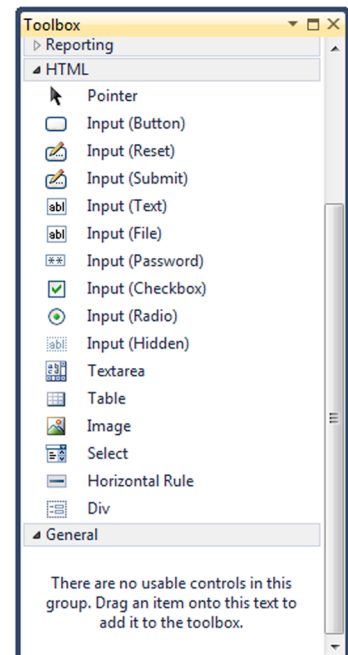


Figura 61. Controles HTML

Figura 62. Código .aspx de un control html



```
Client Objects & Events
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAppWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <input id="Text1" type="text" />
        </div>
    </form>
</body>
</html>
```

Para convertir el elemento HTML en un control de servidor, únicamente tenemos que añadir el atributo `runat = server`.

Figura 63. Código .aspx de un control html



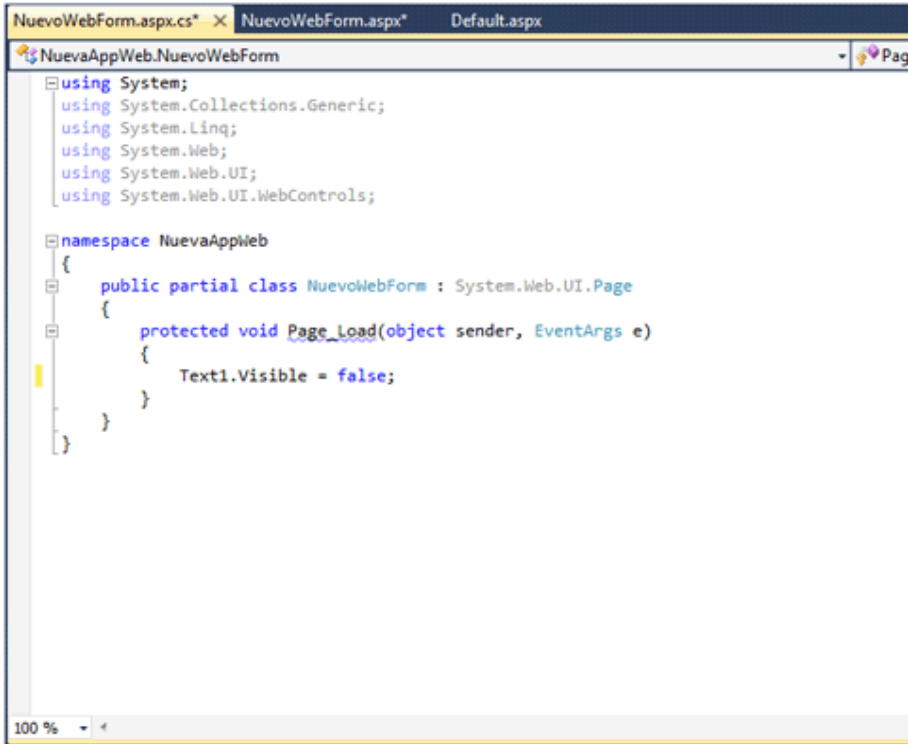
```
Client Objects & Events (Nc
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAppWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <input id="Text1" type="text" runat="server" />
        </div>
    </form>
</body>
</html>
```

Desde este momento podremos acceder desde el archivo de código (.cs) mediante su `id` y modificar sus propiedades.

Figura 64. Código .cs de un control html



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

namespace NuevaAppWeb
{
    public partial class NuevoWebForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Text1.Visible = false;
        }
    }
}
```

Si ejecutamos la aplicación, podremos ver que nuestro elemento no es visible, ya que hemos aplicado la propiedad `Visible = false`.

4.7.2. Controles web

Los controles de servidor HTML proporcionan una forma relativamente rápida de migrar html a ASP.NET, pero no es necesariamente la mejor. Por un lado, los controles HTML y sus atributos no siempre tienen unos nombres intuitivos, y tareas como estilar el control pueden resultar tediosas.

Para resolver estos problemas, ASP.NET pone a nuestra disposición los controles web que básicamente generan el mismo HTML que los controles de servidor HTML, pero disponen de una configuración mucho más sencilla.

Los controles web están disponibles en el Toolbox dentro del apartado *Standard* (figura 65).

Si arrastramos, por ejemplo, el control web Button y lo soltamos en el editor de código, veremos que, como con los controles HTML, se genera un bloque de código por defecto.

Figura 66. Código .aspx de un control web



```
Client Objects & Events (No Ev)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAppWeb.NuevoWebForm" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Button ID="Button1" runat="server" Text="Button" />
</div>
</form>
</body>
</html>
```

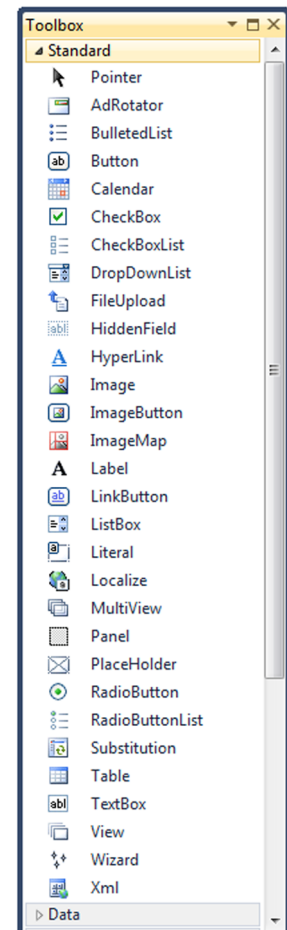


Figura 65. Controles web

Si modificamos una de las propiedades, por ejemplo *Height*, y ejecutamos el sitio web, veremos cómo automáticamente se aplica el valor que hemos indicado a la altura del botón.

Figura 67. Código .cs de un control web

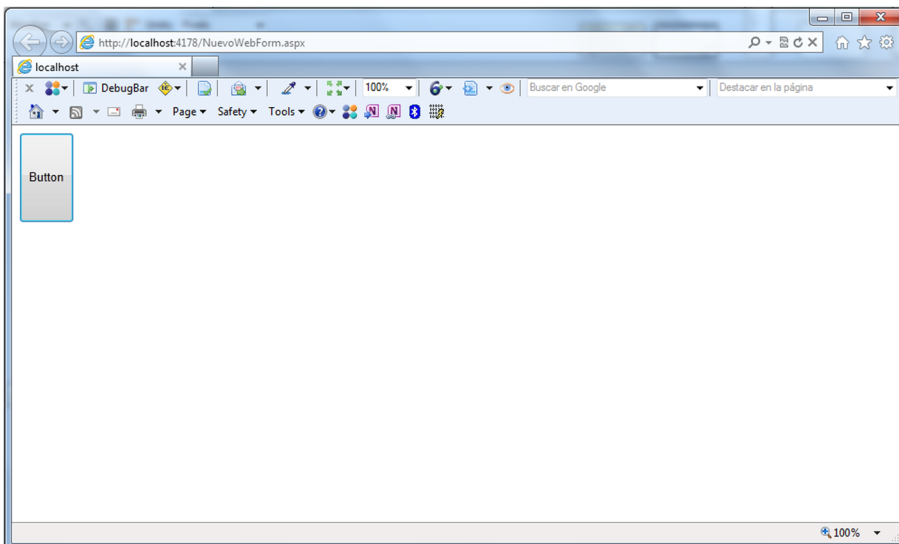


```
Client Objects & Events (No Ever)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAppWeb.NuevoWebForm" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" Text="Button" Height="100"/>
        </div>
    </form>
</body>
</html>
```

Figura 68. Ejecución del ejemplo de un control web



Como hemos podido observar, en el ejemplo de los controles web podemos acceder a las propiedades desde el diseñador de páginas, mientras que en el ejemplo de los controles HTML lo hacíamos desde el archivo adjunto .cs. Esta es otra de las ventajas de utilizar los controles web en lugar de los controles HTML.

4.7.3. Controles Rich

Los controles Rich o *Rich controls* nos aportan complejas interfaces de usuario. Estos controles nos ofrecen funcionalidades mucho más amplias que los controles HTML o web. En realidad cada control Rich es un conjunto de controles web que utilizamos como uno solo. El ejemplo más claro es el del control Calendar que está compuesto por Buttons, Labels o Hyperlinks.

Los controles Rich los encontramos en el apartado *Standard* del Toolbox. Concretamente algunos de los controles Rich más importantes son:

- Calendar: calendario que permite navegar entre meses, días o años.
- Multiview y View: permite implementar diferentes vistas en una misma página y navegar entre ellas.
- AdRotator: *banner* para mostrar una serie de imágenes.

De nuevo arrastraremos un control de este tipo, concretamente un Calendar, y lo soltaremos en el diseñador. Vemos que, aunque internamente está compuesto por varios controles web, únicamente se genera una etiqueta Calendar (figura 69). Hacemos lo mismo con un control web Label, que nos servirá para mostrar el día seleccionado.

Figura 69. Código .aspx de un control Rich



```
Client Objects & Events (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAppWeb.NuevoWebForm" %>

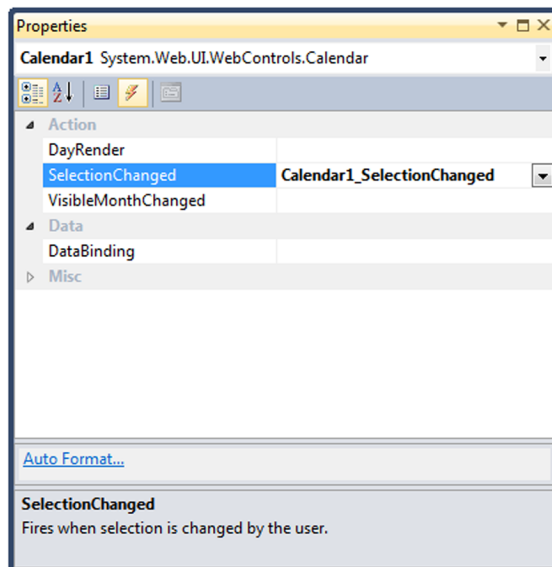
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
      <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </div>
  </form>
</body>
</html>
```

Al igual que los controles web, podemos modificar propiedades o añadir eventos tanto desde el diseñador como desde el archivo adjunto .cs.

A continuación, en el diseñador, en la ventana de propiedades desde el apartado de eventos hacemos doble clic sobre el evento `SelectionChanged`. Esto nos generará automáticamente un evento que se disparará al cambiar el día seleccionado.

Figura 70. Ventana de propiedades del control



Dentro de la función `SelectionChanged` escribimos el código que nos actualizará el control web `Label` con la fecha seleccionada.

Figura 71. Código .cs de un control Rich

```
NuevoWebForm.aspx.cs* x NuevoWebForm.aspx*
NuevaAppWeb.NuevoWebForm - Calendario

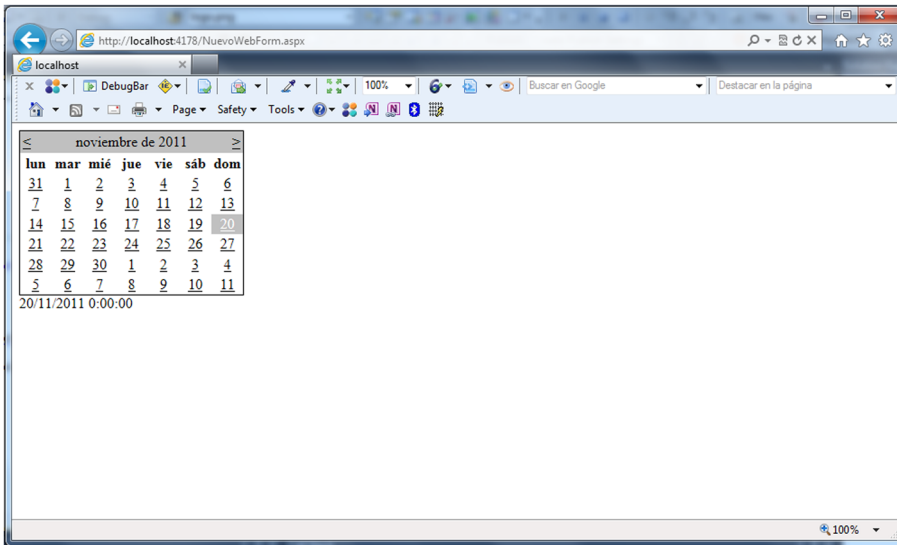
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

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

        protected void Calendar1_SelectionChanged(object sender, EventArgs e)
        {
            Label1.Text = Calendar1.SelectedDate.ToString();
        }
    }
}
```

Ejecutamos y comprobamos que mientras vamos cambiando la selección, la etiqueta se va actualizando correctamente.

Figura 72. Ejecución del ejemplo de un control Rich



4.7.4. Controles de validación

Una de las funcionalidades más comunes de las páginas web es recoger datos insertados por los usuarios. A menudo una página web pide datos a un usuario, que finalmente serán guardados en una base de datos. En la mayoría de casos estos datos deben ser validados para asegurar que no se almacena información incoherente. Idealmente, el dato entrado por el usuario debe ser validado en cliente, mediante JavaScript, y en servidor, en nuestro caso mediante ASP.NET.

Para no tener que escribir el código de validación manualmente, que sería una tarea bastante laboriosa, ASP.NET dispone de seis controles de validación:

- `RequiredFieldValidator`: chequea si el control relacionado está vacío cuando se envía el formulario.
- `RangeValidator`: comprueba que la entrada del usuario esté dentro de un determinado rango.
- `CompareValidator`: chequea que el valor del control cumpla una determinada condición (mayor que, menor que, igual a).
- `RegularExpressionValidator`: comprueba que el valor de un determinado control cumpla una expresión regular.
- `CustomValidator`: permite al desarrollador especificar una función cliente y una de servidor, que validarán la entrada del usuario.

- `ValidationSummary`: proporciona una región con los errores de todos los validadores de la página.

Estos controles realizan la mayor parte del trabajo y el programador solo tiene que configurarlos para su correcto funcionamiento. Podemos encontrar todos estos controles en el Toolbox dentro de la sección *Validation* (figura 73).

Para probar cómo funciona un control de validación, primero debemos insertar el control en la página, así que arrastraremos un `TextBox` desde el Toolbox hasta el editor.

Figura 74. Código .aspx de un control de validación



```
Client Objects & Events (No Events)
Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
  Inherits="NuevaAppWeb.NuevoWebForm"
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
  </form>
</body>
</html>
```

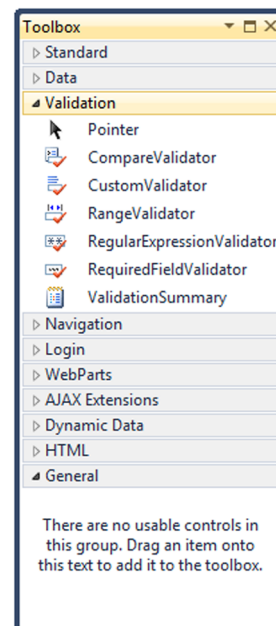


Figura 73. Controles de validación

A continuación arrastramos el control de validación que queremos aplicar, por ejemplo, un `RangeValidator`, y configuramos los siguientes atributos:

- `ErrorMessage`: error que aparecerá cuando la validación falle.
- `MaximumValue`: valor máximo que se puede introducir.
- `MinimumValue`: valor mínimo que se puede introducir.
- `ControlToValidate`: ID del control sobre el cual se aplicará la validación.

Por último arrastramos un `Button` a nuestra página, con el cual realizaremos el envío del formulario.

Figura 75. Código .aspx de un control de validación



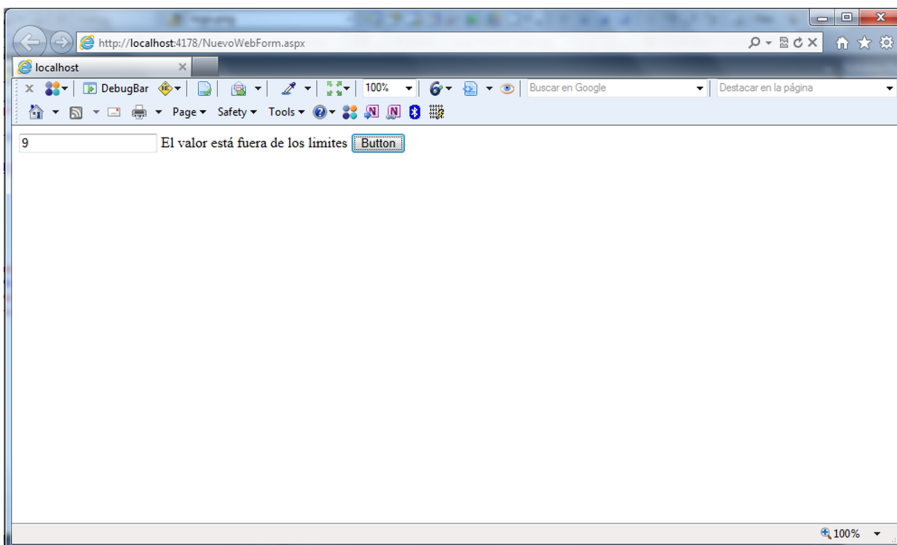
```
Client Objects & Events (No Events)
Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
Inherits="NuevaAppWeb.NuevoWebForm"

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:RangeValidator ID="RangeValidator1" runat="server" ErrorMessage="El valor está fuera de los límites"
MaximumValue="50" MinimumValue="10" ControlToValidate="TextBox1"></asp:RangeValidator>
<asp:Button ID="Button1" runat="server" Text="Button" />
</div>
</form>
</body>
</html>
```

Si ejecutamos nuestra página e introducimos valores en el TextBox, veremos cómo nos aparece el mensaje “El valor está fuera de los límites” cuando enviamos el formulario con un valor erróneo.

Figura 76. Ejecución del ejemplo de un control de validación



4.7.5. Controles de datos

Los controles de datos nos permiten mostrar grandes cantidades de datos con soporte de características de alto nivel, como ordenación, paginación, etc. El control por excelencia y más utilizado de este grupo es el GridView.

Este control es extremadamente flexible y nos muestra los datos en forma de filas y columnas. Con este control se cubren las funcionalidades más comunes con el manejo de datos, como son la paginación, la ordenación o la selección de elementos.

Ejemplo práctico de uso del GridView

Para saber mejor cómo funciona el GridView, veremos un pequeño ejemplo práctico.

Al igual que los demás controles, encontraremos el GridView disponible en el Toolbox (figura 77). Arrastramos un GridView al editor y vemos cómo automáticamente genera el código xml.

Figura 78. Código .aspx de un control de datos

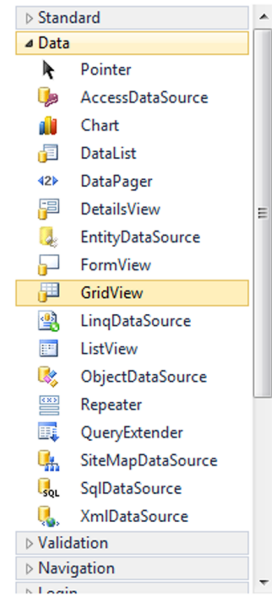
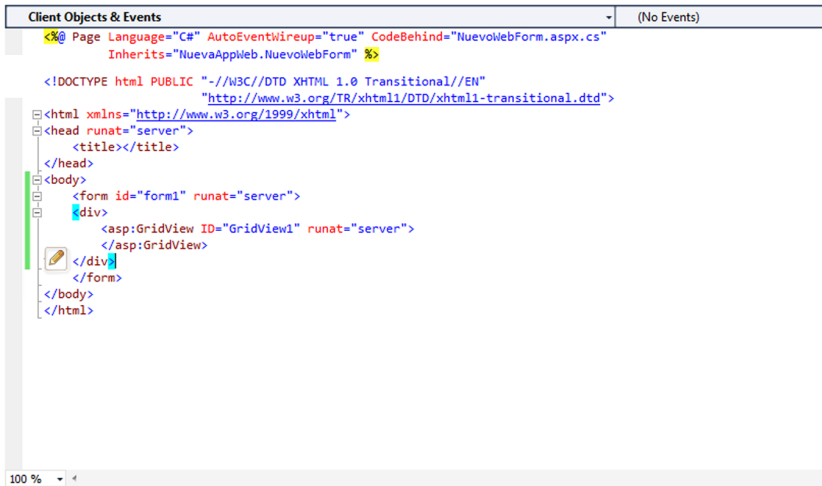


Figura 77. Controles de datos

A continuación debemos proporcionar el origen de los datos al control. La forma más sencilla es construir una lista de objetos, así que creamos una clase llamada `Persona`.

Figura 79. Ejemplo de un GridView

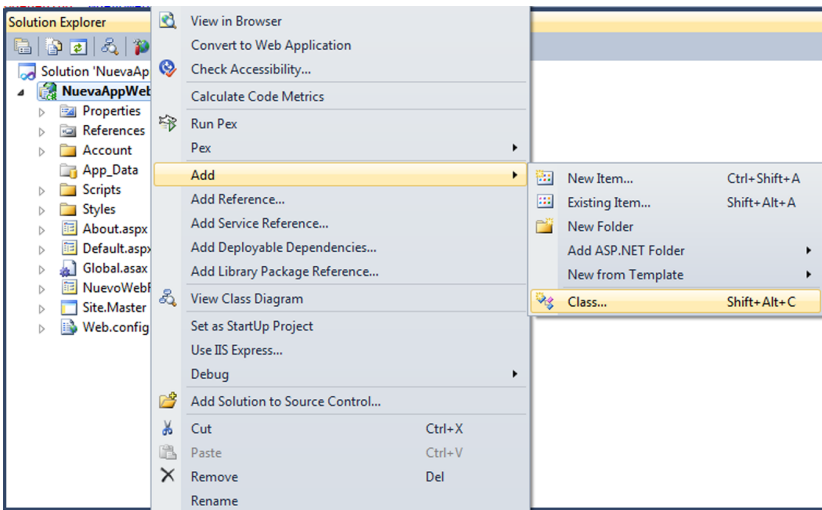


Figura 80. Ejemplo de un GridView

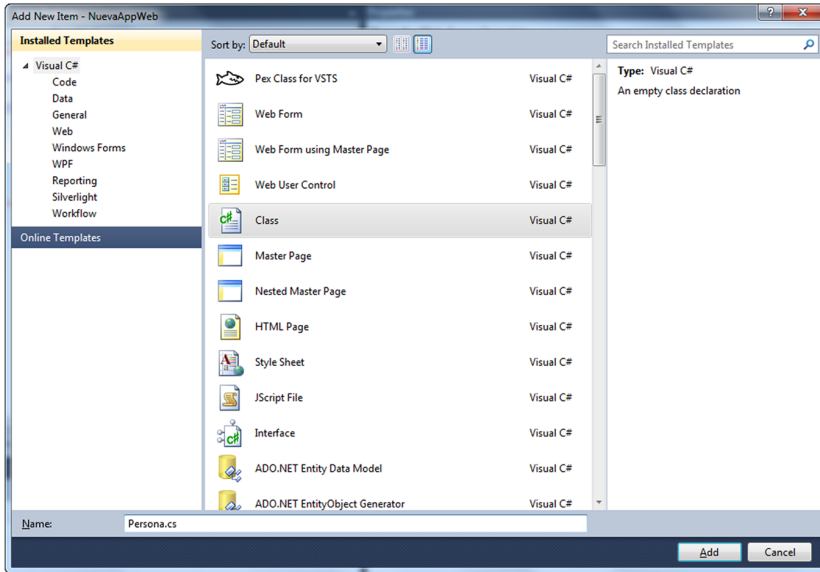
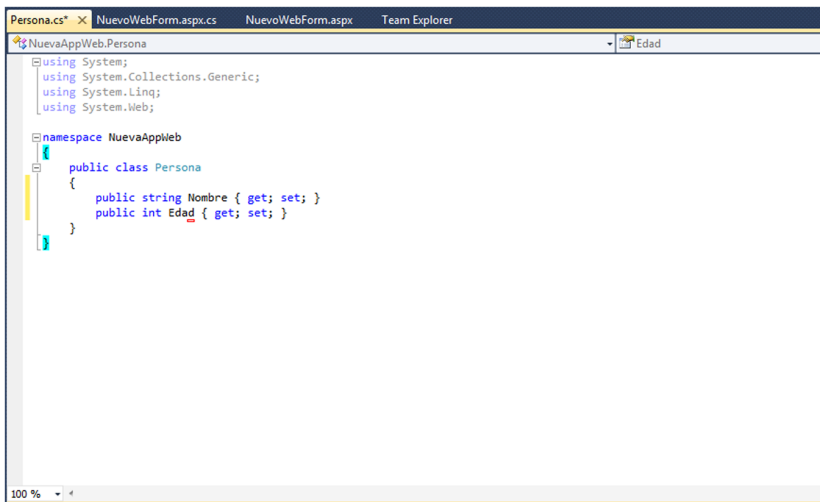


Figura 81. Ejemplo de un GridView



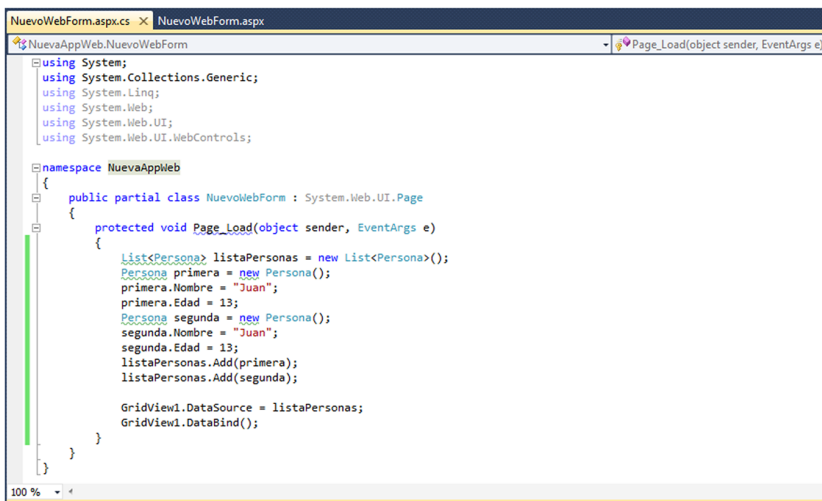
A continuación, en el evento Page_Load del archivo .cs generamos una lista de objetos Persona.

Figura 82. Ejemplo de un GridView



Finalmente, relacionamos la lista anterior creada con el origen de datos o `DataSource` del `GridView`.

Figura 83. Ejemplo de un `GridView`



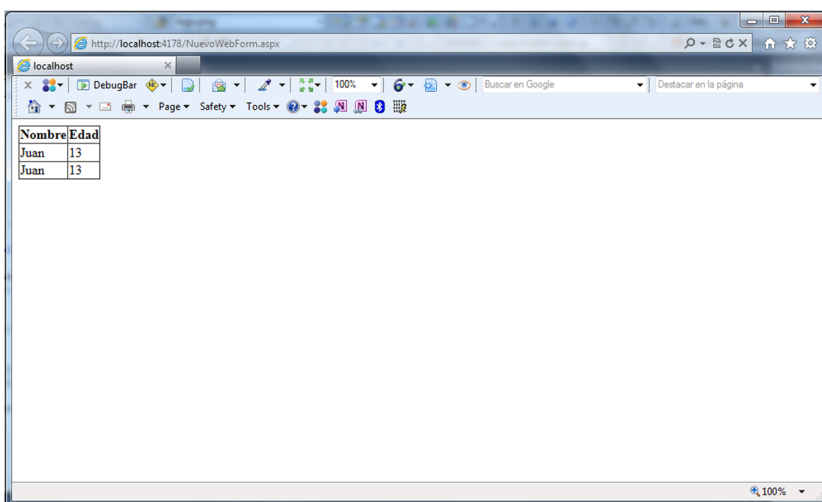
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace NuevaAppWeb
{
    public partial class NuewaAppWeb : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            List<Persona> listaPersonas = new List<Persona>();
            Persona primera = new Persona();
            primera.Nombre = "Juan";
            primera.Edad = 13;
            Persona segunda = new Persona();
            segunda.Nombre = "Juan";
            segunda.Edad = 13;
            listaPersonas.Add(primera);
            listaPersonas.Add(segunda);

            GridView1.DataSource = listaPersonas;
            GridView1.DataBind();
        }
    }
}
```

Si ejecutamos nuestro web site podemos ver cómo automáticamente se crea una `Grid`, donde las columnas son las propiedades de la clase `Persona` y las filas corresponden a cada uno de los objetos de la lista.

Figura 84. Ejemplo de una `GridView`



4.8. Controles de usuario

Una aplicación web bien construida divide el código en diferentes bloques independientes. Cuanto más modular es la aplicación, más fácil es mantener su código, solucionar errores y reutilizar componentes. Para la organización del código en diferentes componentes, ASP.NET pone a disposición del desarrollador los `UserControl` o controles de usuario. Los `UserControl` se parecen mucho a los `WebForm`, ya que están compuestos por una porción de código con etiquetas HTML y controles de servidor (el archivo `.ascx`) y un archivo de código con la lógica y los eventos.

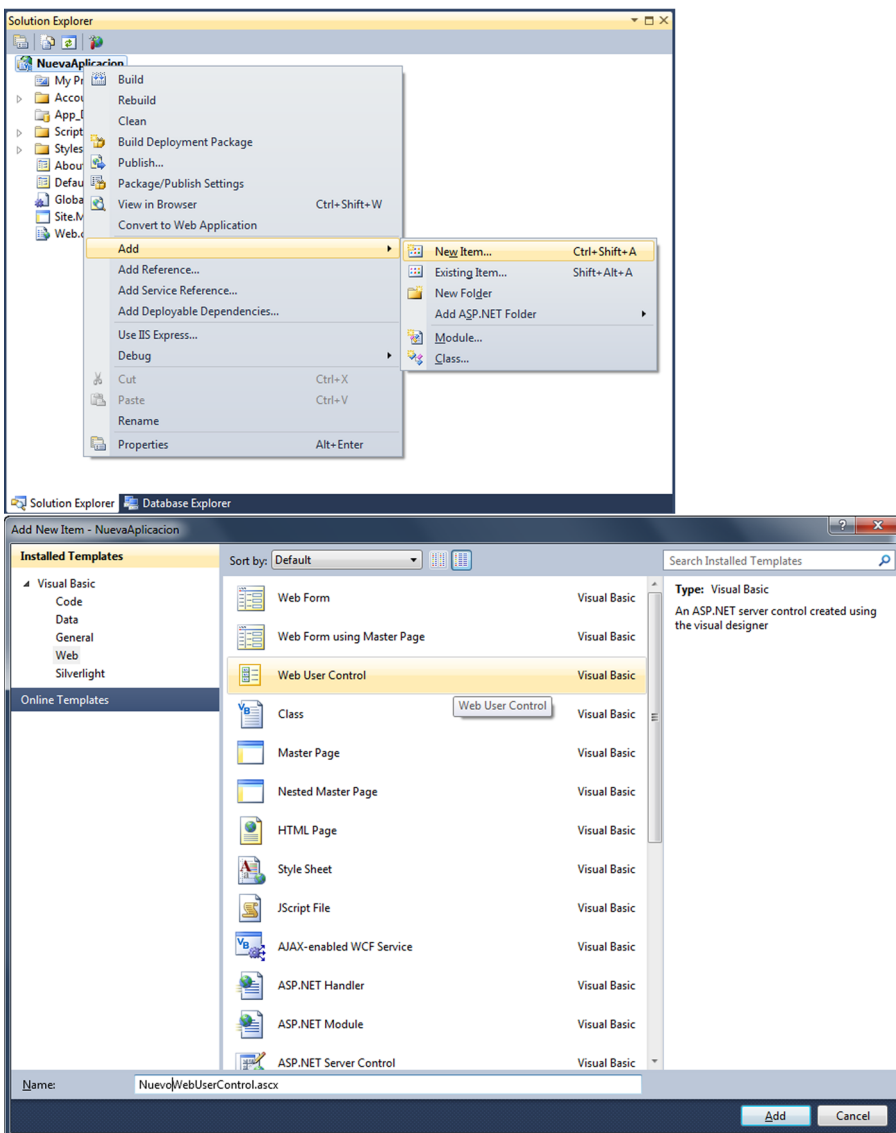
Las únicas diferencias entre los `UserControl` y los `WebForm` són:

- a) Los UserControl utilizan la extensión `.ascx` en lugar de `.aspx` y la clase del código `.cs` hereda de `UserControl` en lugar de `Page`.
- b) El archivo `.ascx` comienza con la directiva `<%@Control%>` en lugar de `<%@Page%>`.
- c) Los UserControl no pueden ser solicitados directamente por un navegador. Siempre deben estar embebidos dentro de un WebForm.

4.8.1. Creando un UserControl


La manera de crear un UserControl en Visual Studio es muy similar a la forma de crear un WebForm. Debemos seleccionar *Add > New item* y escoger *Web User Control* de la lista. Le damos el nombre de, por ejemplo, `NuevoControl.ascx`.

Figura 85. Creando un UserControl



Modificamos el archivo `.ascx` para que quede de la siguiente forma:

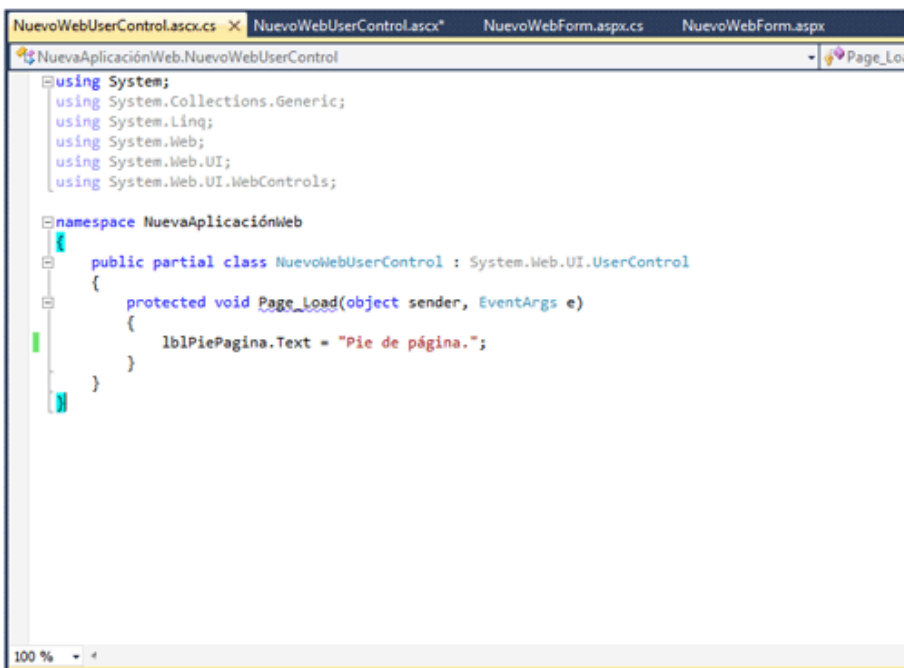
Figura 86. Código .aspx de un UserControl



```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebUserControl.ascx.cs"
    Inherits="NuevaAplicaciónWeb.NuevoWebUserControl" %>
<asp:Label ID="lblPiePagina" runat="server" />
```

En el archivo de código relacionado vemos cómo los eventos son muy parecidos a los de los WebForm. Modificamos también este archivo para que quede de la siguiente manera:

Figura 87. Código .cs de un UserControl



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

namespace NuevaAplicaciónWeb
{
    public partial class NuevoWebUserControl : System.Web.UI.UserControl
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            lblPiePagina.Text = "Pie de página.";
        }
    }
}
```

Para poder probar este UserControl es necesario insertarlo en un WebForm. Se trata de un proceso de dos pasos. En primer lugar necesitamos agregar la directiva Register a la página que contendrá el UserControl justo después de la directiva Page.

```
<%@ Page Language = "C#" AutoEventWireup = "true" CodeBehind = "NuevoWebForm.aspx.cs"
    Inherits = "NuevaAplicacionWeb.NuevoWebForm" %>
<%@ Register TagPrefix = "customControls" TagName = "pie" Src = "NuevoWebUserControl.ascx" %>
```

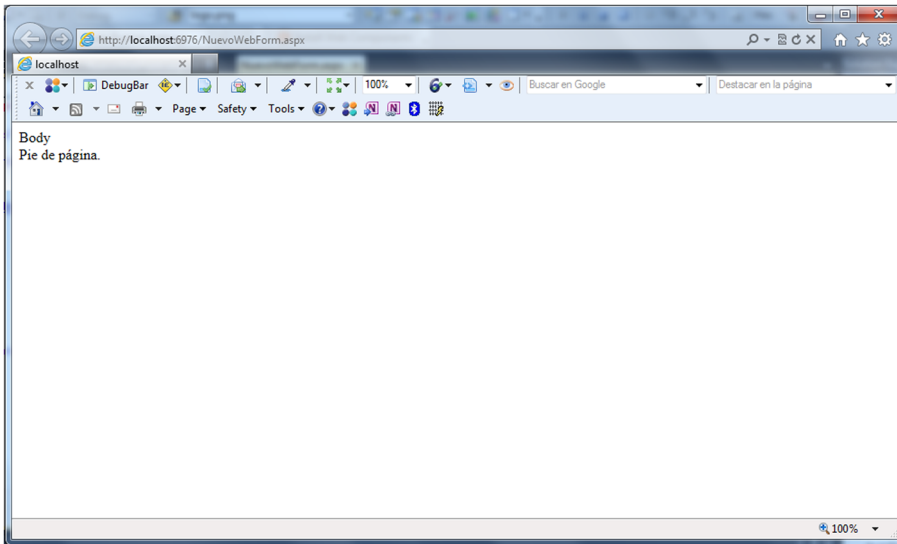
En esta directiva, como podemos ver, se debe especificar un prefijo y un nombre. La directiva `src` sirve para informar de la localización del UserControl.

A continuación debemos insertar el UserControl en el WebForm como si fuera un control de servidor más mediante la siguiente directiva, especificando el prefijo y el nombre:

```
<div>
  <customControls:pie runat = "server"></customControls:pie>
</div>
```

Si ejecutamos la página podremos ver cómo el UserControl se muestra como si fuera un control más (figura 88).

Figura 88. Ejecución del ejemplo del UserControl



4.9. Páginas maestras

Cuando diseñamos un sitio web, normalmente disponemos de una serie de elementos comunes en todas las páginas: el logotipo de la organización, un pie con información, una serie de enlaces generales, etc.

Una novedad que introdujo ASP.NET son las páginas maestras. Gracias a estas se reduce el tiempo de diseño y el mantenimiento del sitio.

Una **página maestra** es una combinación de código HTML y controles de servidor, como cualquier otra página de ASP.NET, con la peculiaridad de que se almacena en un archivo con extensión `.master` en lugar de la habitual extensión `.aspx`.

Otra diferencia respecto a los WebForm se centra en la cabecera del archivo `.master`.

En una página maestra, la directiva `Page` es sustituida por la directiva `Master`.

```
<%@ Master Language = "C#" AutoEventWireup = "true" CodeBehind = "Site1.master.cs"
    Inherits = "NuevaAppWeb.Site1" %>
```

Como hemos dicho, en una página maestra podemos introducir contenido HTML, componentes de servidor y cualquier otro contenido que pueda existir en una página ASP.NET. Lo que distinguirá a la página maestra de otras es la aparición como parte del contenido de uno o más componentes `ContentPlaceHolder`.

Figura 89. Código `.aspx` de una página maestra



```
Client Objects & Events (No Events)
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs"
    Inherits="NuevaAppWeb.Site1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
<asp:ContentPlaceHolder ID="head" runat="server">
</asp:ContentPlaceHolder>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
</div>
</form>
</body>
</html>
```

4.9.1. Componentes `ContentPlaceHolder` y `Content`

Al diseñar una página maestra iremos dejando unos huecos o contenedores, representados por componentes `ContentPlaceHolder`. Después, en cada página individual se introducirá el contenido específico aportado por los componentes `Content`.

El componente `ContentPlaceHolder`, por tanto, aparecerá en el archivo `.master`. La propiedad que más nos interesa de este componente es el `ID`, a la que asignaremos un identificador o nombre que será el que sirva para enlazarlo con el control `Content`.

Figura 90. Código .aspx de una página maestra



```
Client Objects & Events (No Events)
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs"
    Inherits="NuevaAppWeb.Site1" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
<asp:ContentPlaceHolder ID="head" runat="server">
</asp:ContentPlaceHolder>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
</div>
</form>
</body>
</html>
```

Figura 91. Código .aspx de una página de contenido



```
Client Objects & Events (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAppWeb.NuevoWebForm" MasterPageFile="~/Site1.Master" %>

<asp:Content ID="ContentHead" ContentPlaceHolderID="head" runat="server">
</asp:Content>

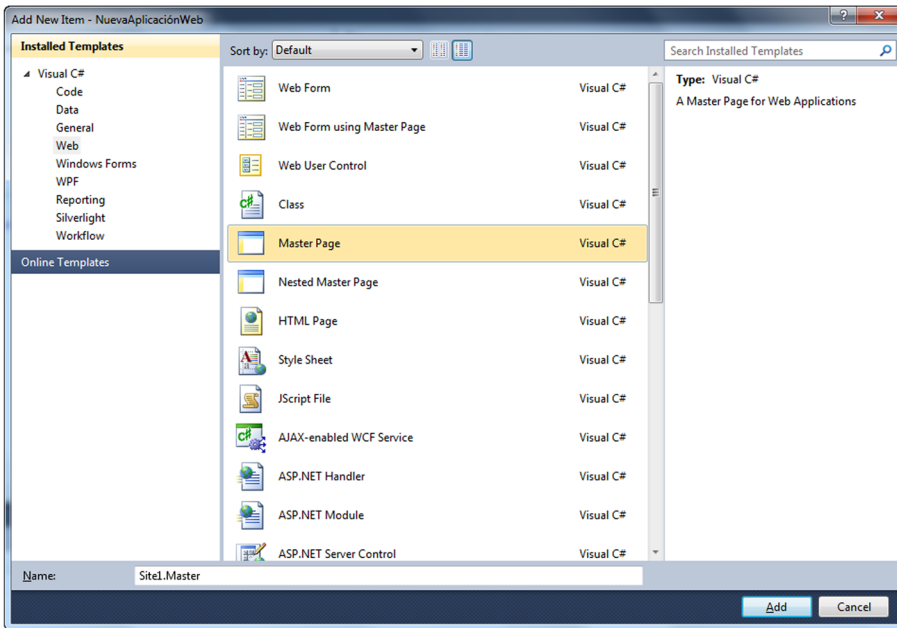
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
</asp:Content>
```

Vemos que para referenciar la página maestra en el WebForm específico se utiliza la propiedad `MasterPageFile`, que aparece como atributo en la directiva `Page` de aquellas páginas donde interese incluir el diseño base.

4.9.2. Ejemplo práctico

Ahora que conocemos a grandes rasgos los elementos principales de las páginas maestras, podemos ver cómo utilizarlas en el diseño de un sitio sencillo. Comenzaremos creando un nuevo proyecto web vacío. A continuación abriremos el cuadro de dialogo *Add > New item* con el botón derecho del ratón sobre nuestro proyecto y seleccionamos *Master Page* (figura 92).

Figura 92. Añadiendo plantilla de MasterPage



Esta página maestra por defecto contendrá únicamente un contenedor `ContentPlaceholder`.

Figura 93. Código .aspx de una página maestra



Introducimos un contenido por defecto en el `ContentPlaceholder`, que se mostrará siempre y cuando no lo sobrescribamos con el componente `Content` que veremos a continuación, y agregamos los elementos propios de la página maestra, como la cabecera o el pie.

Figura 94. Código .aspx de una página maestra

```

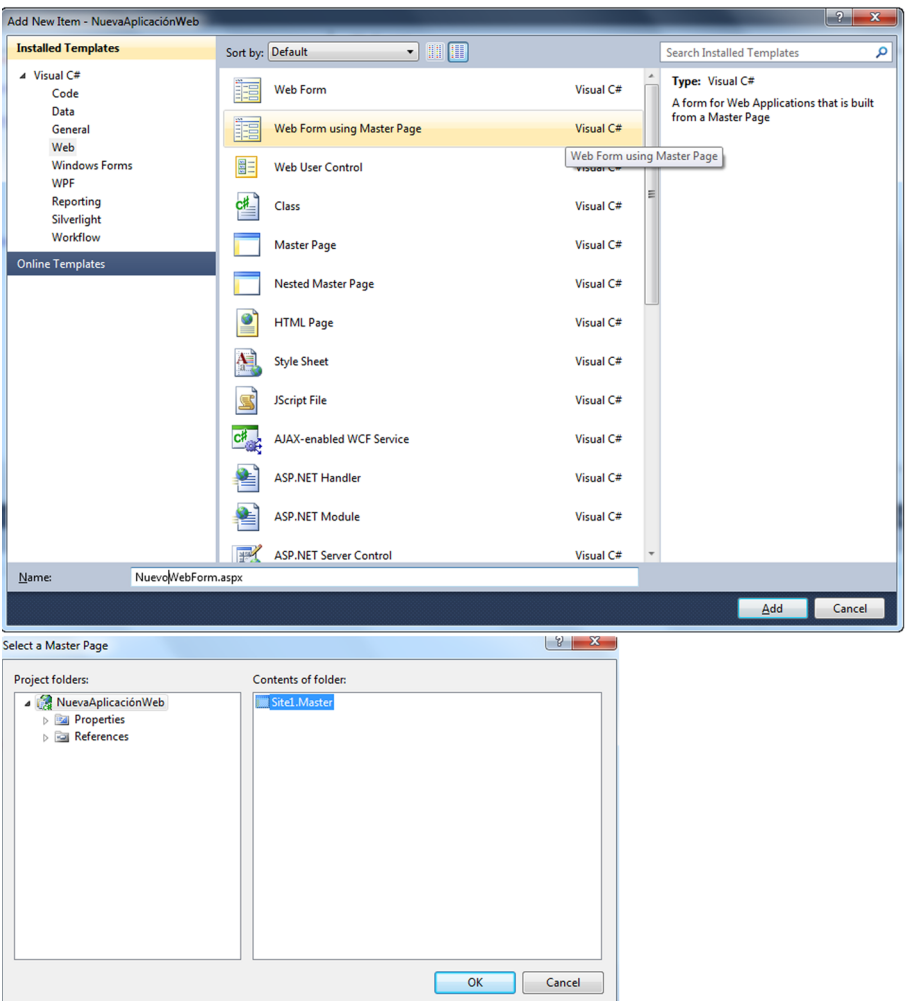
Client Objects & Events (No Events)
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs"
    Inherits="NuevaAplicaciónWeb.Site1" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>Cabecera</div>
<div>
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
    Contenido por defecto
</asp:ContentPlaceHolder>
</div>
<div>Pie</div>
</form>
</body>
</html>
    
```

A continuación abrimos otra vez el menú *Add > New item* y seleccionamos *Web Forms using Master Page*. Se abrirá una ventana mostrando en la lista derecha las páginas maestras disponibles. Elegimos la que acabamos de crear.

Figura 95. Añadiendo plantilla de página de contenido



Para cada página que añadamos al proyecto y que enlacemos a la página maestra, tendremos que agregar un contenido en el correspondiente objeto `Content`. Para relacionar el componente `Content` con su correspondiente `ContentPlaceHolder`, únicamente debemos indicar en el primero el ID del segundo.

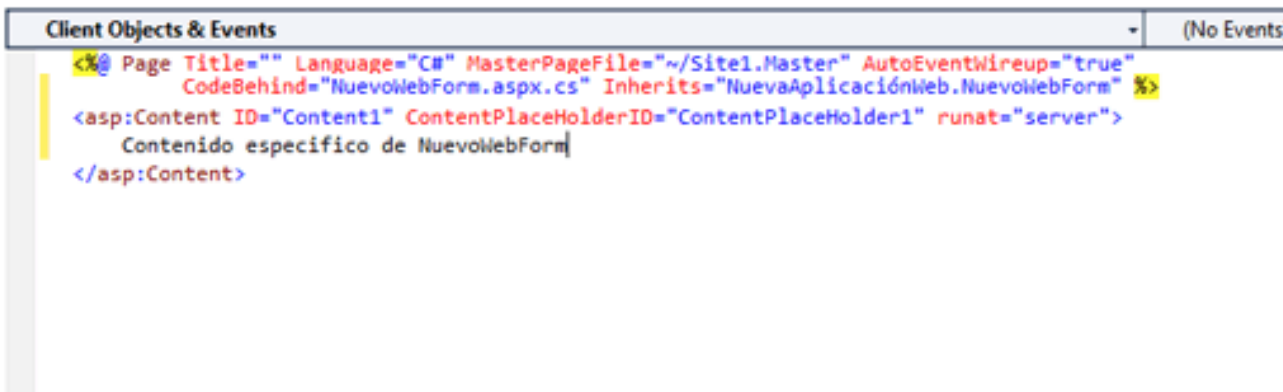
Figura 96. Código `.aspx` de una página de contenido



```
Client Objects & Events (No Events)
<%@ Page Title="" Language="C#" MasterPageFile="~/Site1.Master" AutoEventWireup="true"
CodeBehind="NuevoWebForm.aspx.cs" Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceholder1" runat="server">
</asp:Content>
```

Introducimos el contenido que queremos que aparezca en el espacio del `ContentPlaceHolder`.

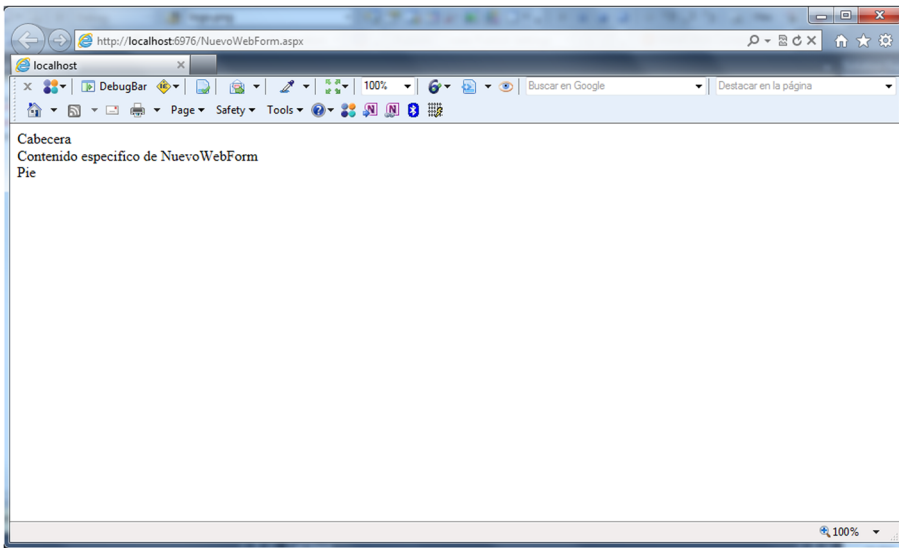
Figura 97. Código `.aspx` de una página de contenido



```
Client Objects & Events (No Events)
<%@ Page Title="" Language="C#" MasterPageFile="~/Site1.Master" AutoEventWireup="true"
CodeBehind="NuevoWebForm.aspx.cs" Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceholder1" runat="server">
    Contenido específico de NuevoWebForm
</asp:Content>
```

Con estos simples pasos ya tendremos una página accesible desde el navegador con una cabecera, un contenido y un pie. Si ejecutamos la aplicación, veremos la página que hemos creado incluida dentro de la página maestra (figura 98).

Figura 98. Ejecución del ejemplo de MasterPage



5. ASP.NET AJAX

En otros módulos hemos visto cómo implementar la técnica AJAX mediante JavaScript, y más concretamente mediante el objeto `XMLHttpRequest` que existe en todos los navegadores actuales. Esta forma de construir peticiones asíncronas, como hemos visto, es bastante laboriosa e implica invertir mucho tiempo de trabajo.

ASP.NET pone a disposición del desarrollador ASP.NET AJAX para facilitar la tarea de la implementación AJAX en nuestros sitios web. ASP.NET AJAX consta de dos partes principales: una parte del lado del cliente y una parte del lado del servidor.

La parte del lado del cliente es un conjunto de bibliotecas JavaScript. Básicamente, estas bibliotecas añaden funcionalidades a JavaScript para facilitar el desarrollo de nuestras aplicaciones.

La parte de servidor de ASP.NET AJAX, que es la que de verdad nos interesa, funciona en un nivel superior. Incluye controles y componentes que utilizan las bibliotecas JavaScript del lado del cliente sin que tengamos que tener conocimiento de ellas.

5.1. El ScriptManager

El ScriptManager es el cerebro del modelo de servidor ASP.NET AJAX. Se trata de un control web que no tiene ningún aspecto visual; sin embargo, realiza la tarea clave de conectar los controles de servidor AJAX con las bibliotecas JavaScript.

Para agregar el ScriptManager en una página, lo podemos encontrar en el Toolbox en el apartado *Ajax Extensions* (figura 99). Una vez arrastrado y soltado en el editor, obtendremos el siguiente código (figura 100):

Figura 100. Código .aspx de un ScriptManager



```

Client Objects & Events (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<div>
</div>
</form>
</body>
</html>
  
```

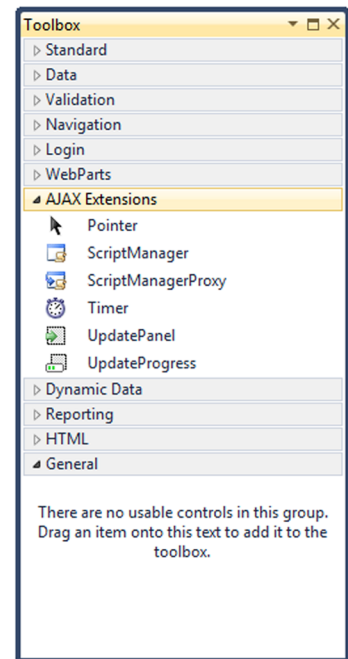


Figura 99. Caja de herramientas

Cada página que utiliza controles ASP.NET AJAX requiere de una instancia y solo una de ScriptManager.

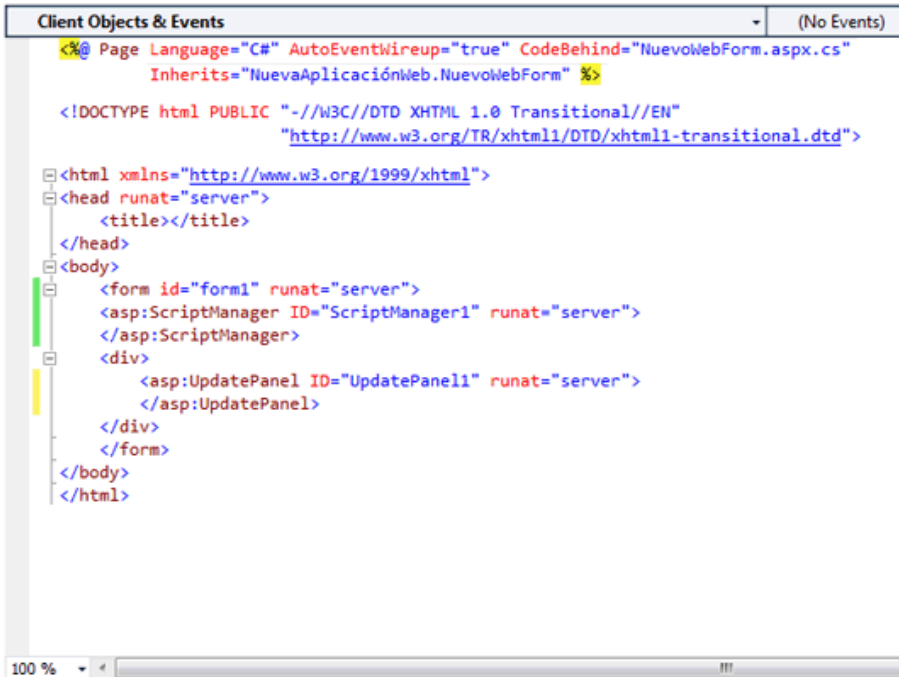
5.2. Renderizaciones parciales con UpdatePanel

Sin duda alguna, el control AJAX más importante de todos es el UpdatePanel. Este control nos permite hacer renderizaciones parciales de nuestra página, es decir, podemos refrescar partes de nuestra página sin tener que recargar la página entera. El escenario habitual de un UpdatePanel es el siguiente:

- El usuario lanza un evento, como el clic de un botón dentro de un UpdatePanel.
- ASP.NET AJAX intercepta el evento y realiza la petición asíncrona de la página al servidor.
- En el servidor se procesa la página de forma normal y se devuelve al navegador.
- El navegador recibe la página y el código JavaScript se encarga de refrescar solo el UpdatePanel donde se encontraba el botón.

El UpdatePanel trabaja en conjunto con el control ScriptManager y, por lo tanto, cuando utilizamos UpdatePanel debe haber antes un ScriptManager. Por lo que para realizar un ejemplo, primero insertamos un ScriptManager en nuestra página. A continuación, en el mismo apartado *Ajax Extensions* del Toolbox encontraremos el UpdatePanel. Al insertarlo en nuestra página, se generará el siguiente código (figura 101):

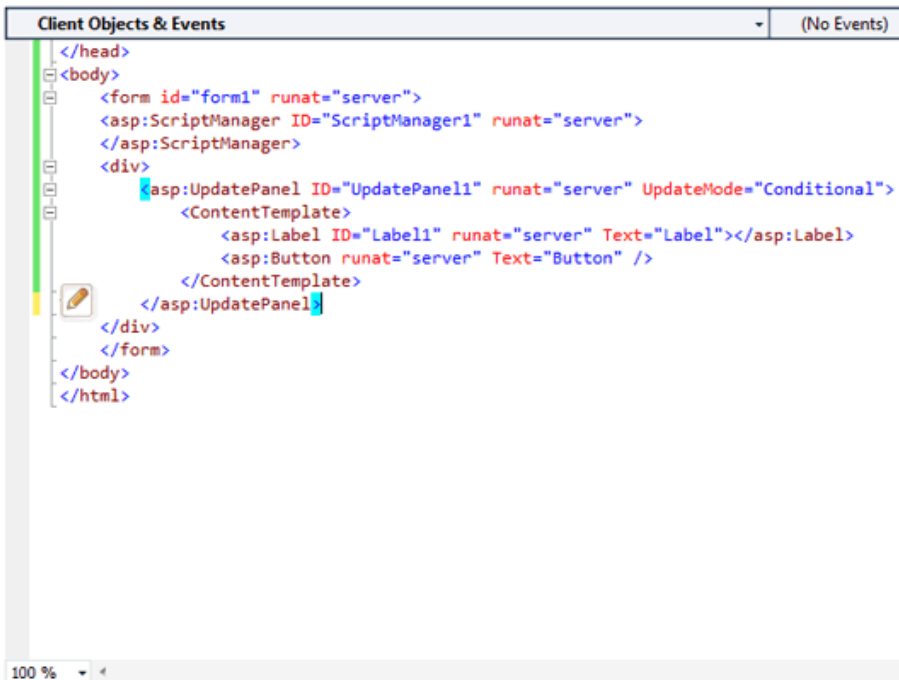
Figura 101. Código .aspx de un UpdatePanel



```
Client Objects & Events (No Events)
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<div>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
</asp:UpdatePanel>
</div>
</form>
</body>
</html>
```

A continuación, insertamos una Label dentro del UpdatePanel para ir mostrando el resultado de la llamada asíncrona y un Button para ir lanzando dicha llamada.


Figura 102. Código .aspx de un UpdatePanel



```
Client Objects & Events (No Events)
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
        <ContentTemplate>
          <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
          <asp:Button runat="server" Text="Button" />
        </ContentTemplate>
      </asp:UpdatePanel>
    </div>
  </form>
</body>
</html>
```

Realizamos el mismo proceso para agregar dos UpdatePanel más con un Label y un Button cada uno.

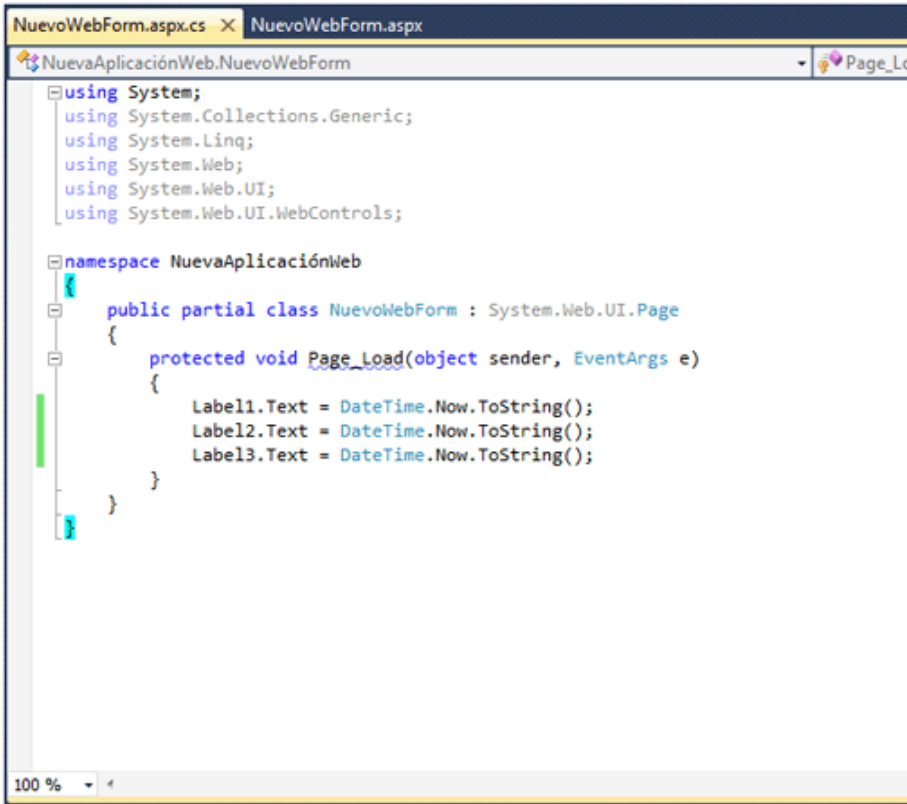
Figura 103. Código .aspx de un UpdatePanel



```
Client Objects & Events (No Events)
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
        <ContentTemplate>
          <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
          <asp:Button runat="server" Text="Button" />
        </ContentTemplate>
      </asp:UpdatePanel>
      <asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional">
        <ContentTemplate>
          <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
          <asp:Button runat="server" Text="Button" />
        </ContentTemplate>
      </asp:UpdatePanel>
      <asp:UpdatePanel ID="UpdatePanel3" runat="server" UpdateMode="Conditional">
        <ContentTemplate>
          <asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>
          <asp:Button runat="server" Text="Button" />
        </ContentTemplate>
      </asp:UpdatePanel>
    </div>
  </form>
</body>
```

En el archivo .cs de la página escribimos el siguiente código para actualizar cada uno de los Labels:

Figura 104. Código .cs de un UpdatePanel

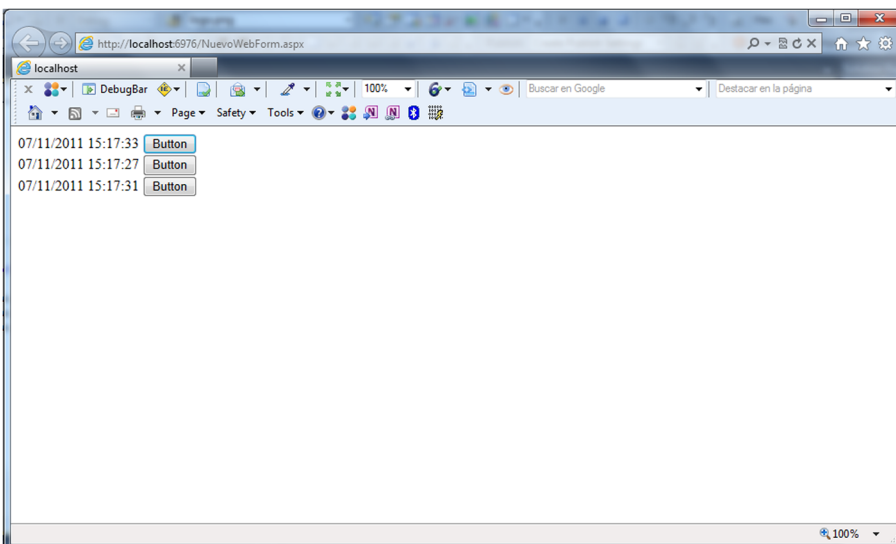


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace NuevaAplicaciónWeb
{
    public partial class NuevoWebForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Label1.Text = DateTime.Now.ToString();
            Label2.Text = DateTime.Now.ToString();
            Label3.Text = DateTime.Now.ToString();
        }
    }
}
```

Si ejecutamos la página veremos cómo para cada clic de cualquier Button solo se refresca el UpdatePanel donde está contenido, aunque hayamos variado los tres Labels.

Figura 105. Ejecución del ejemplo de UpdatePanel



5.3. Indicación de estado con UpdateProgress

Mientras un navegador renueva una página, muestra al usuario algún tipo de indicación, por ejemplo un icono animado o un cambio progresivo de color en la barra de direcciones, que permite saber que se encuentra ocupado, esperando recibir respuesta del servidor. Por defecto, las aplicaciones AJAX no

cuentan con ese tipo de comunicación de retorno, que resulta importante, ya que sin ella la persona que utiliza el navegador no sabe si la comunicación está en curso o es que la aplicación sencillamente ha dejado de funcionar.

Introducir una indicación de estado en una aplicación ASP.NET AJAX resulta muy sencillo, ya que existe un componente llamado `UpdateProgress`, que se encarga de colocar en la página el elemento de notificación que se desee. Esa notificación, además, puede mantenerse oculta hasta el momento en que el tiempo de espera supere un cierto límite, y vuelve a desaparecer automáticamente en cuanto la respuesta haya llegado. Las dos propiedades clave de este componente son `AssociatedUpdatePanelID` y `DisplayAfter`. Con la primera, el `UpdateProgress` se asocia con un `UpdatePanel` de la página, mientras que la segunda establece el número de milisegundos de espera tras los que se hará visible la indicación.

Al igual que otros componentes, `UpdateProgress` es un contenedor inicialmente vacío. Para que realice su trabajo, es necesario introducir en el mismo algún contenido: una etiqueta de texto, un pequeño gráfico, etc.

Partiendo del ejemplo del subapartado anterior, añadiremos en el interior del primer `UpdatePanel` un `UpdateProgress` y dentro de este, una simple etiqueta de texto con un mensaje.

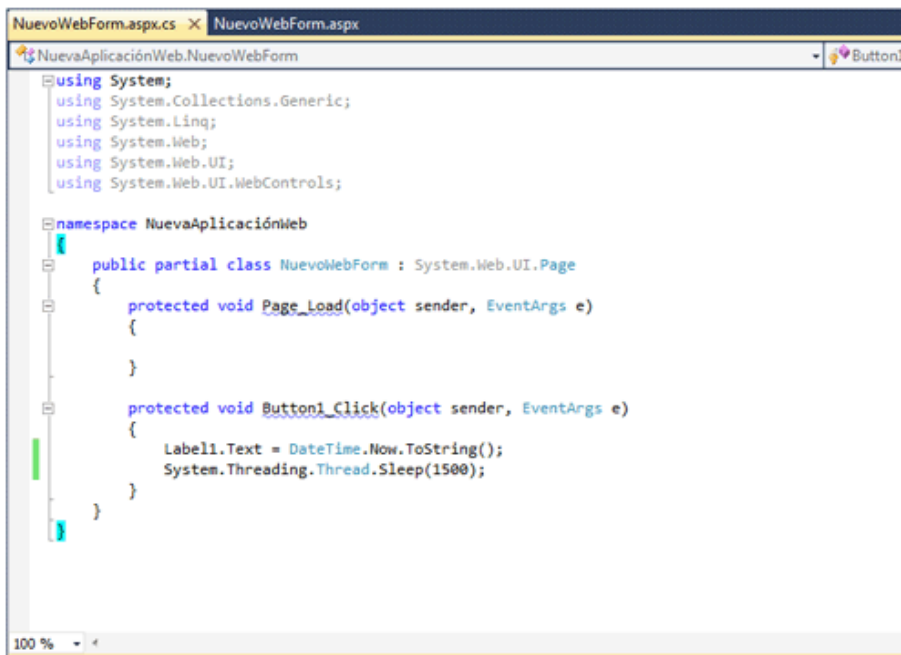
Figura 106. Código .aspx de un `UpdateProgress`

The image shows a screenshot of a code editor window titled "Client Objects & Events" with a "(No Events)" tab. The code is for an ASP.NET page. It starts with a page directive: `<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs" Inherits="NuevaAplicaciónWeb.NuevoWebForm" %>`. The document type is `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`. The root element is `<html xmlns="http://www.w3.org/1999/xhtml">`. Inside the `<head>` block, there is a `<title></title>` tag. The `<body>` block contains a `<form id="form1" runat="server">` tag. Inside the form, there is an `<asp:ScriptManager ID="ScriptManager1" runat="server">` tag. Below the script manager is a `<div>` tag containing an `<asp:UpdatePanel ID="UpdatePanel1" runat="server">` tag. Inside the update panel, there is a `<ContentTemplate>` block. Inside the content template, there is an `<asp:Label ID="Label1" runat="server" />` tag, followed by an `<asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click"/>` tag, and then an `<asp:UpdateProgress ID="UpdateProgress1" runat="server">` tag. Inside the update progress tag, there is a `<ProgressTemplate>` block containing an `<asp:Label ID="Label2" runat="server" Text="Cargando..."/>` tag. The code ends with `</ProgressTemplate>`, `</asp:UpdateProgress>`, `</ContentTemplate>`, `</asp:UpdatePanel>`, `</div>`, `</form>`, `</body>`, and `</html>`. The code editor shows a vertical scrollbar on the left and a zoom level of 100% at the bottom left.

Si ejecutamos el proyecto, no advertiremos en principio ninguna diferencia, ya que el `UpdateProgress` está programado para mostrar el mensaje si la respuesta tarda más de medio segundo en llegar. En un entorno real podría darse el caso de que la petición tardara más de medio segundo, pero probando en local, la respuesta es casi inmediata.

Para simular ese tiempo de espera escribiremos el siguiente código que retarda el hilo de ejecución:

Figura 107. Código .cs de un UpdateProgress



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

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

        protected void Button1_Click(object sender, EventArgs e)
        {
            Label1.Text = DateTime.Now.ToString();
            System.Threading.Thread.Sleep(1500);
        }
    }
}
```

Ahora, al ejecutar de nuevo la aplicación, se podrá apreciar perfectamente el funcionamiento del componente UpdateProgress.

5.4. ASP.NET AJAX Control Toolkit

Los controles UpdatePanel y UpdateProgress son bastante útiles. Sin embargo, son los únicos controles AJAX que se encuentran en ASP.NET. Para ampliar la gama de controles AJAX, Microsoft y la comunidad ASP.NET desarrollaron ASP.NET AJAX Control Toolkit.

ASP.NET AJAX Control Toolkit está formado por decenas de controles que utilizan las bibliotecas JavaScript de ASP.NET AJAX para crear efectos sofisticados. Dos factores que ASP.NET AJAX Control Toolkit tiene a su favor son:

- Es completamente gratuito.
- Se incluye el código fuente completo, gracias a lo cual el programador podrá personalizar los controles que lo componen.

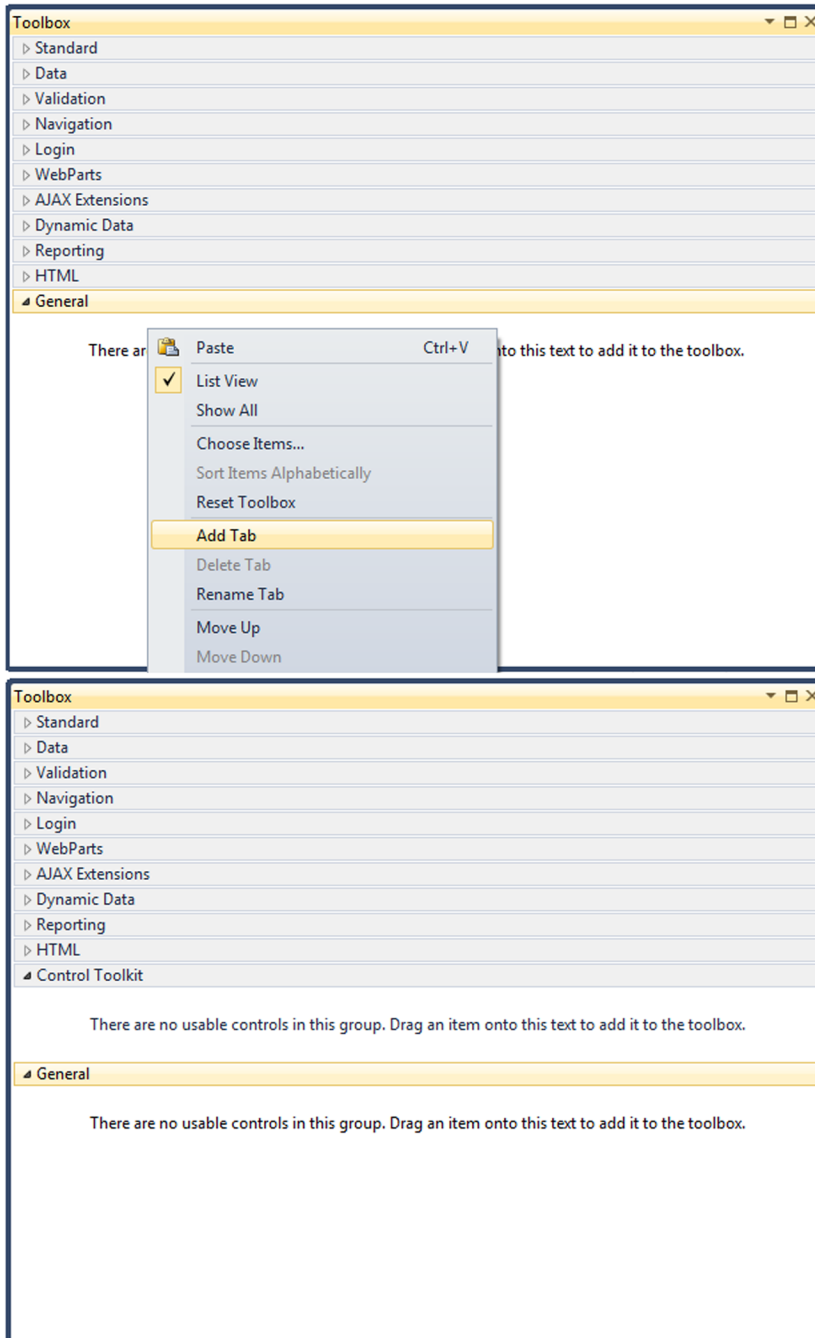
5.4.1. Instalando ASP.NET AJAX Control Toolkit

Para descargar ASP.NET AJAX Control Toolkit, iremos a la dirección Ajax Control Toolkit.

Una vez hayamos descargado el archivo .zip únicamente descomprimos el archivo `AjaxControlToolkit.dll`. Para conseguir integrar ASP.NET AJAX Control Toolkit con nuestro sitio web, debemos seguir una serie de pasos:

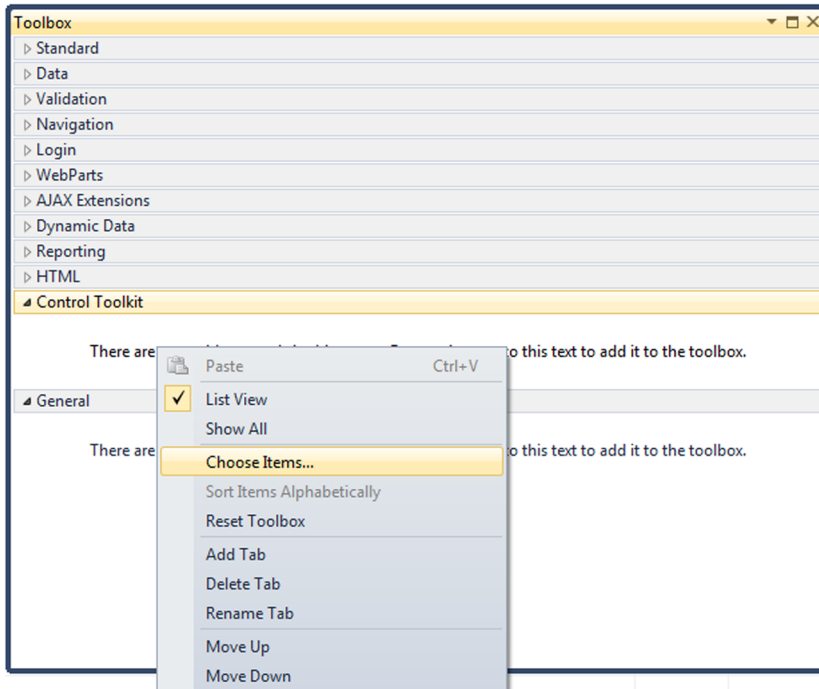
1) Primero es necesario crear un apartado nuevo en nuestro Toolbox pulsando con el botón derecho sobre el mismo y seleccionando *Add Tab* (figura 108). Al nuevo apartado lo llamamos “Control Toolkit”.

Figura 108. Añadiendo sección para Control Toolkit



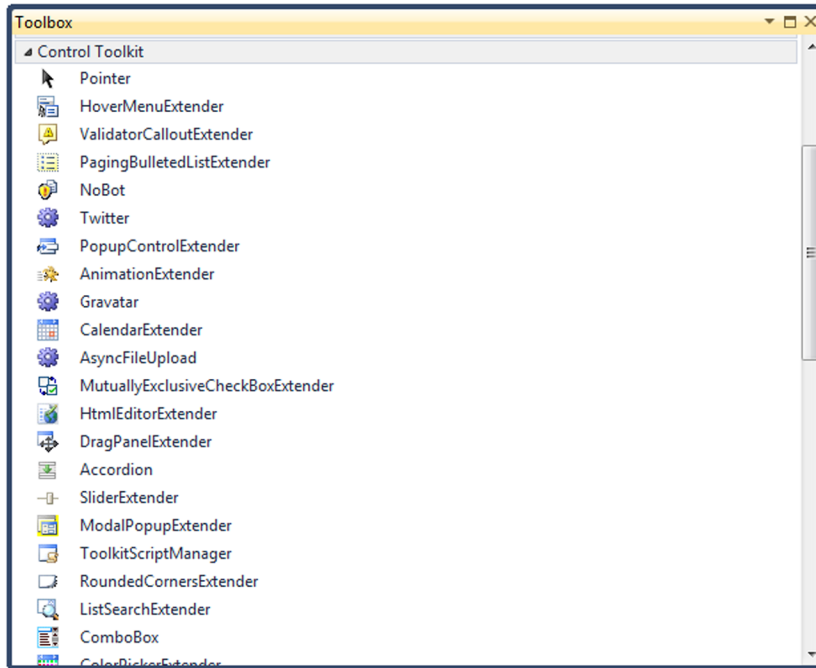
En este momento tenemos un nuevo apartado en el Toolbox, pero no contiene ningún control. Para añadir los controles pulsamos con el botón derecho sobre *Control Toolkit* y seleccionamos *Choose items* (figura 109). A continuación, exploramos en nuestro sistema y escogemos el archivo `AjaxControlToolkit.dll` que anteriormente extrajimos.

Figura 109. Añadiendo sección para Control Toolkit



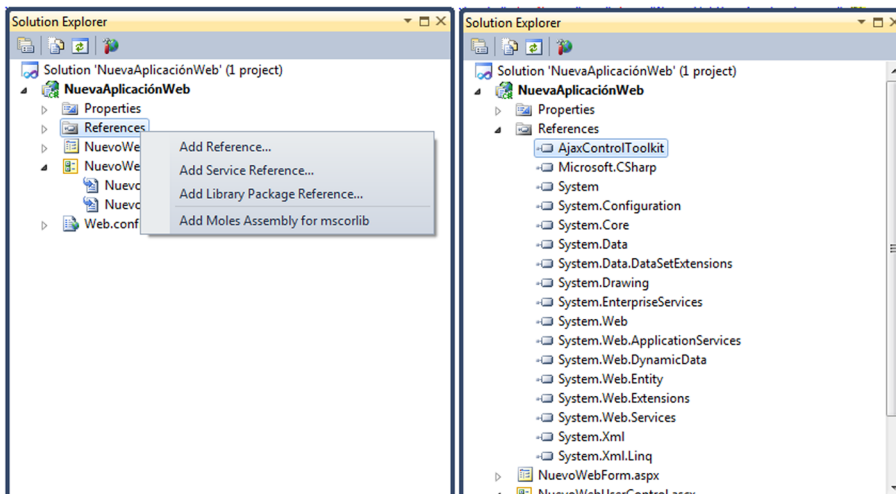
Automáticamente, se crearán todos los controles en nuestro nuevo apartado Control Toolkit (figura 110).

Figura 110. Añadiendo sección para Control Toolkit



2) El segundo paso importante es añadir la biblioteca `AjaxControlToolkit.dll` a nuestro sitio web. Pulsamos con el botón derecho sobre la carpeta `References` y seleccionamos `Add Reference`. A continuación, exploramos en nuestro sistema y escogemos el archivo `AjaxControlToolkit.dll` (figura 111).

Figura 111. Añadiendo referencia a Control Toolkit



Una vez añadida la biblioteca de los controles debemos registrarla en cada WebForm o UserControl que queramos con la directiva `Register`.

```
<% Register Assembly = "AjaxControlToolkit" Namespace = "AjaxControlToolkit" TagPrefix = "asp" %>
```

A partir de ahora podremos utilizar cualquier control de ASP.NET AJAX Control Toolkit mediante la etiqueta `<asp: NombreControl></asp: NombreControl>`.

La biblioteca ASP.NET AJAX Control Toolkit, como ya hemos dicho, dispone de multitud de controles. El objetivo del módulo no se trata de un análisis exhaustivo de todos los controles, sino de ver unos ejemplos básicos lo más genéricos posible. En los dos subapartados siguientes trataremos dos controles de ejemplo para ver cómo utilizarlos: `Accordion` y `AutoCompleteExtender`.

5.4.2. Accordion

Un control `Accordion` es un contenedor en forma de pila con varios paneles donde se puede ver solo uno cada vez. Cada panel tiene una cabecera y un contenido. Cuando se hace clic en la cabecera de uno de ellos, el panel se expande y se muestra el contenido a la vez que los demás paneles se minimizan.

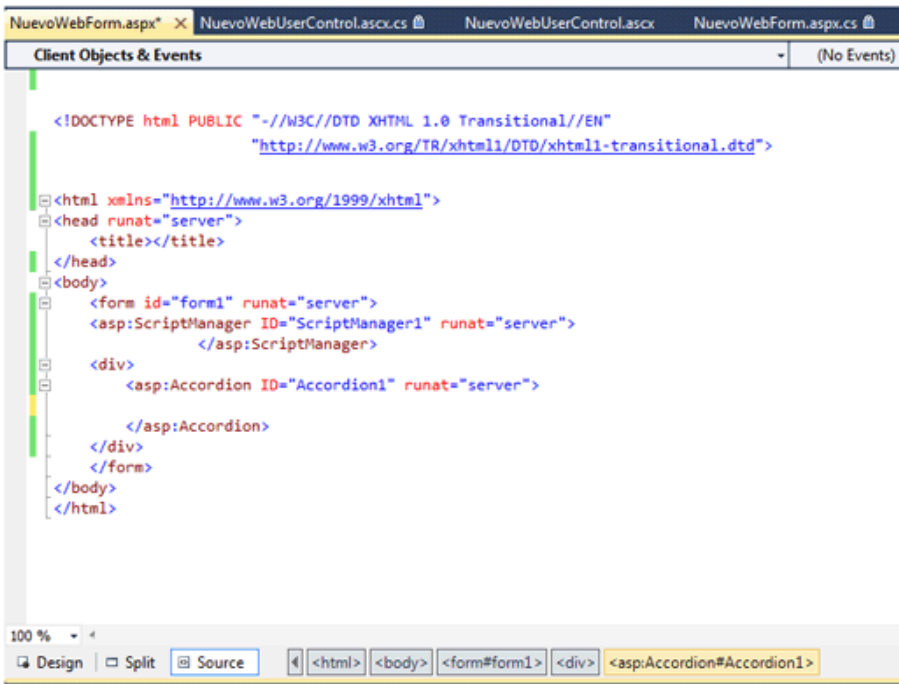
Para realizar un pequeño ejemplo con el control `Accordion` crearemos un nuevo `WebForm`, al que le añadiremos la directiva:

```
<%@ Register Assembly = "AjaxControlToolkit" Namespace = "AjaxControlToolkit"
    TagPrefix = "asp" %>
```

Como cualquier otro control AJAX, los controles de la biblioteca ASP.NET AJAX Control Toolkit necesitan de la presencia de un control `ScriptManager`.

A continuación arrastraremos el control `Accordion` del `Toolbox`, con lo que se creará el código siguiente (figura 112):

Figura 112. Código .aspx de un Accordion



```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<div>
<asp:Accordion ID="Accordion1" runat="server">
</asp:Accordion>
</div>
</form>
</body>
</html>

```

Seguidamente añadiremos el siguiente código para construir nuestro Accordion (figura 113):

Figura 113. Código .aspx de un Accordion



```

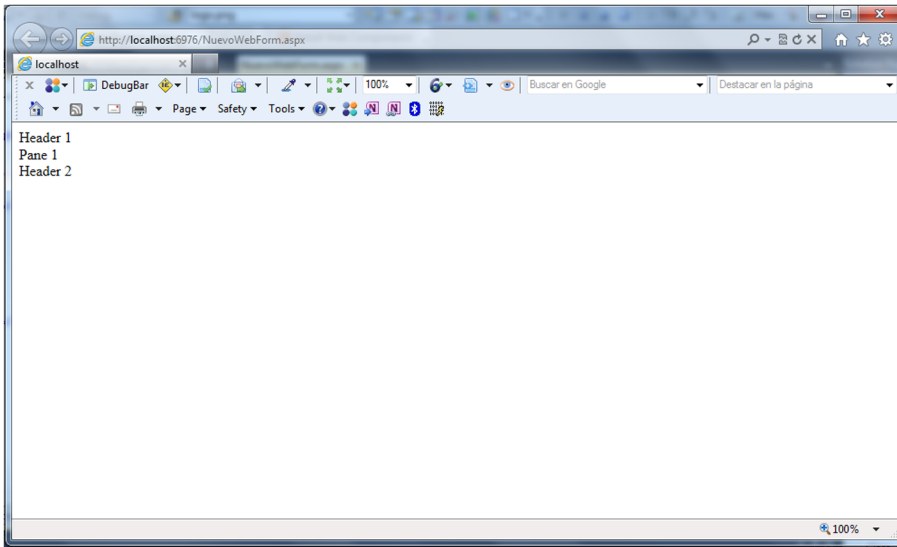
<body>
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<div>
<asp:Accordion ID="Accordion1" runat="server">
<Panes>
<asp:AccordionPane ID="AccordionPane1" runat="server">
<Header>
Header 1
</Header>
<Content>
Pane 1
</Content>
</asp:AccordionPane>
<asp:AccordionPane ID="AccordionPane2" runat="server">
<Header>
Header 2
</Header>
<Content>
Pane 2
</Content>
</asp:AccordionPane>
</Panes>
</asp:Accordion>
</div>
</form>

```

Vemos cómo cada sección de nuestro Accordion se corresponderá con un `AccordionPane` y cada `AccordionPane` contiene un `Header` (que será lo que se muestre como cabecera) y un `Content` (que será lo que se muestre cuando el panel se expanda).

Si ejecutamos nuestra web podremos comprobar el funcionamiento del Accordion (figura 114).

Figura 114. Ejecución del ejemplo del Accordion



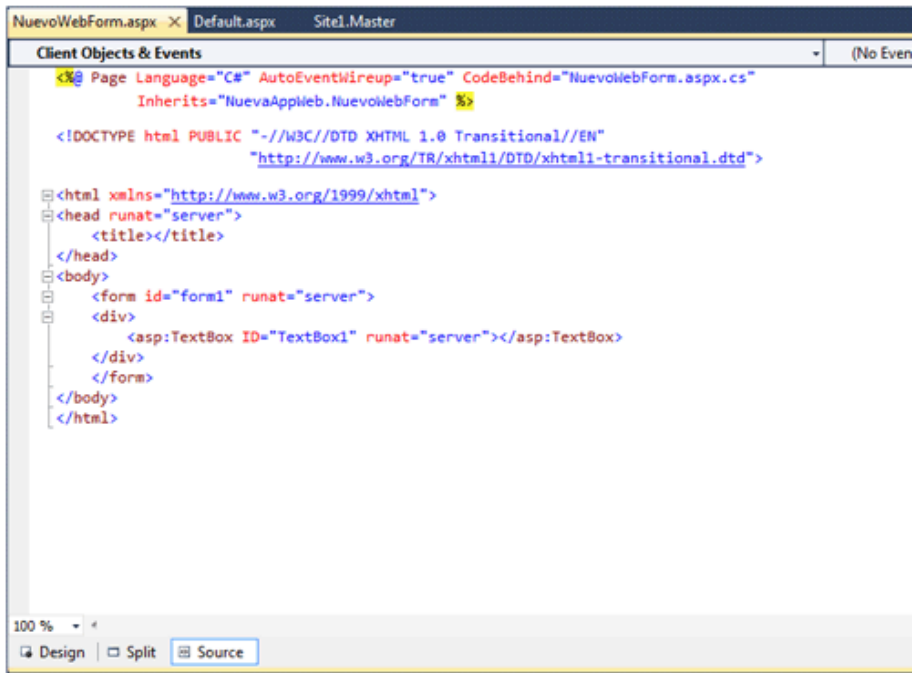
5.4.3. AutoCompleteExtender

El ejemplo anterior del Accordion muestra un control totalmente nuevo que tiene características AJAX. En la biblioteca ASP.NET AJAX, Control Toolkit no es el caso más común, ya que lo más normal son controles que se complementan con los ya existentes en ASP.NET para añadirles funcionalidades AJAX.

Una de estas extensiones de control es el AutoCompleteExtender, que se complementa con controles como el TextBox, y nos va ofreciendo una lista de posibles valores mientras el usuario introduce caracteres. Si el usuario hace clic en uno de los elementos de la lista, el valor se copia dentro del cuadro de texto.

Para ver la funcionalidad del AutoCompleteExtender crearemos un pequeño ejemplo. Primero insertaremos un TextBox en un nuevo WebForm.

Figura 115. Código .aspx de un AutoCompleteExtender



```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
    Inherits="NuevaAppliWeb.NuevoWebForm" %>

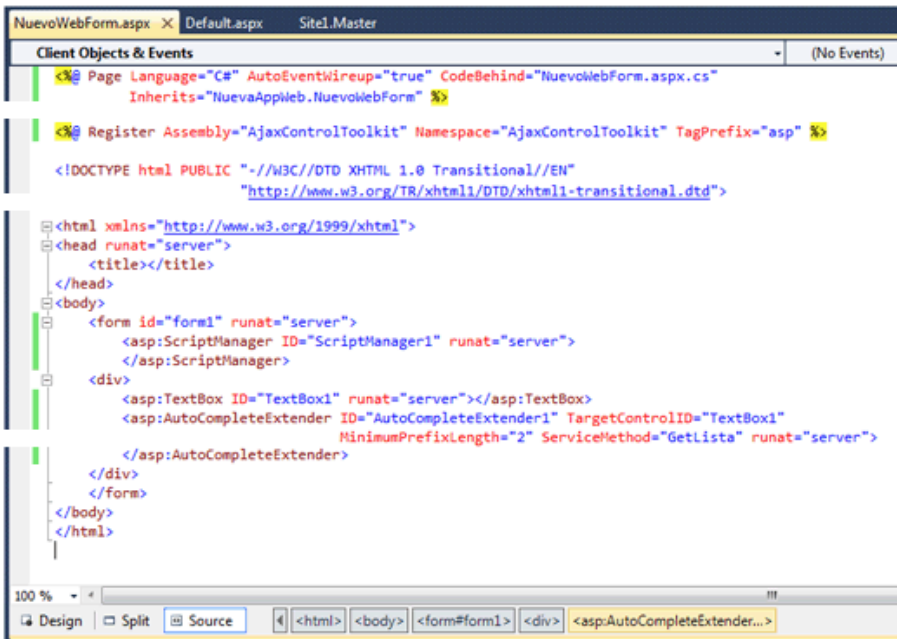
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
  </div>
  </form>
</body>
</html>
```

A continuación, y como en todos los controles AJAX insertaremos un Script-Manager que controlará la parte JavaScript del control. En este momento ya estamos en condiciones de agregar el control `AutoCompleteExtender` a continuación de nuestro `TextBox`. Únicamente debemos configurar tres propiedades del control:

- `TargetControlID`: identificador del control sobre el que se aplicará la extensión.
- `MinimumPrefixLength`: a partir de que el usuario escriba x caracteres, se mostrará la lista.
- `ServiceMethod`: método que se encargará de devolver los ítems que se mostrarán en la lista

Figura 116. Código .aspx de un AutoCompleteExtender



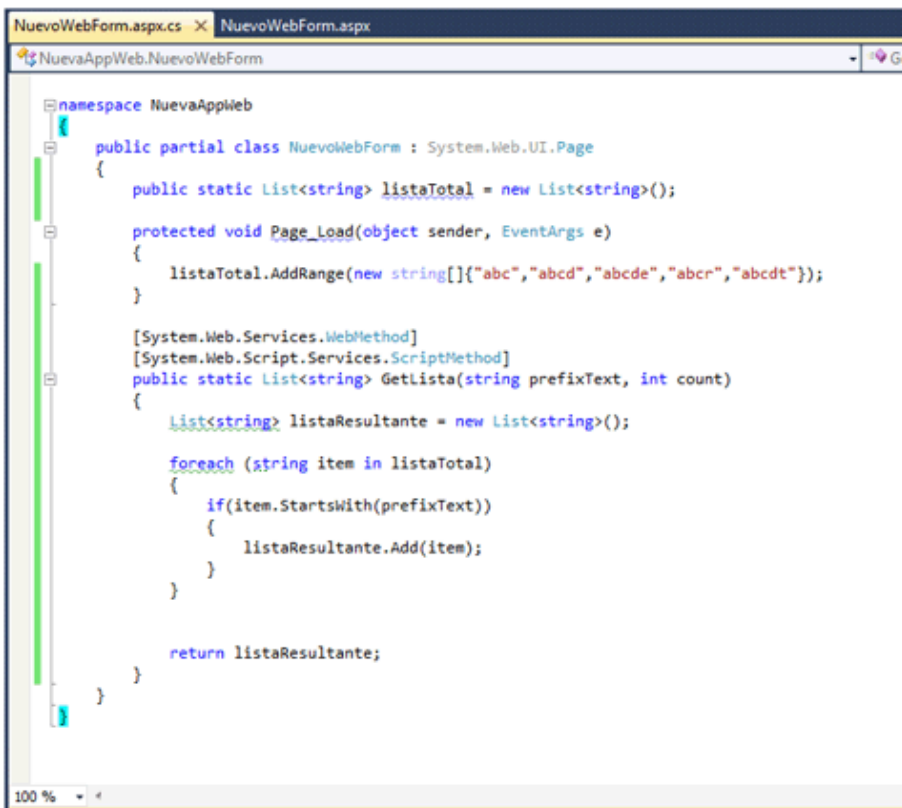
```

NuevoWebForm.aspx X Default.aspx Site1.Master
Client Objects & Events (No Events)
<% Page Language="C#" AutoEventWireup="true" CodeBehind="NuevoWebForm.aspx.cs"
   Inherits="NuevaAppWeb.NuevoWebForm" %>
<% Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<div>
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:AutoCompleteExtender ID="AutoCompleteExtender1" TargetControlID="TextBox1"
   MinimumPrefixLength="2" ServiceMethod="GetLista" runat="server">
</asp:AutoCompleteExtender>
</div>
</form>
</body>
</html>
100 %
Design Split Source <html> <body> <form#form1> <div> <asp:AutoCompleteExtender...>

```

El método que devolverá la lista de *strings* que se mostrará debe tener las anotaciones de `WebMethod` y `ScriptMethod` para que sea accesible directamente desde Javascript.

Figura 117. Código .cs de un AutoCompleteExtender



```

NuevoWebForm.aspx.cs X NuevoWebForm.aspx
NuevaAppWeb.NuevoWebForm
namespace NuevaAppWeb
{
    public partial class NuevoWebForm : System.Web.UI.Page
    {
        public static List<string> listaTotal = new List<string>();

        protected void Page_Load(object sender, EventArgs e)
        {
            listaTotal.AddRange(new string[]{"abc", "abcd", "abcde", "abcr", "abcdt"});
        }

        [System.Web.Services.WebMethod]
        [System.Web.Script.Services.ScriptMethod]
        public static List<string> GetLista(string prefixText, int count)
        {
            List<string> listaResultante = new List<string>();

            foreach (string item in listaTotal)
            {
                if(item.StartsWith(prefixText))
                {
                    listaResultante.Add(item);
                }
            }

            return listaResultante;
        }
    }
}
100 %

```

Vemos cómo tenemos una lista con todos los *string* "listaTotal" y únicamente debemos comprobar en la función `GetLista` si cada elemento de dicha lista empieza por la cadena entrada por el usuario.

Si ejecutamos la aplicación web veremos cómo a medida que el usuario inserta caracteres la lista se va actualizando.

Figura 118. Ejecución del ejemplo del `AutoCompleteExtender`

