



ASP.NET





Manual Practico

Ing. Dennis Antonio Pérez Escobar - 01T
Ing. Emerson Francisco Cartagena Candelario – 01L
Ing. José Mario Menjivar Pineda – 02L

Este manual ha sido elaborado para orientar al estudiante de ASP.NET en el desarrollo de sus prácticas de laboratorios. El uso de este manual debe ser antes, durante y después de la práctica, de tal forma que ofrece un método facilitador en su proceso de enseñanza/aprendizaje durante esta asignatura.

En el desarrollo de esta asignatura se ha designado realizar las prácticas en 16 sesiones semanales de laboratorios, los que incluyen 13 prácticas, 2 parciales prácticos y un proyecto de cátedra, durante los cuales, el estudiante aplicará los conceptos y las técnicas fundamentalmente necesarios para el dominio de lenguaje de programación.

El software a utilizar es el siguiente:

-  Visual Studio 2017.
-  .NET Framework 4.5 o superior
-  SQL 2012 o superior.
-  Bootstrap.

1 | Formularios WEB y Controles de Validación

Objetivos:

- ✓ Crear script básicos con controles WEB
- ✓ Identificar el uso de los controles de validación.

Introducción.

ASP.NET es un Framework creado por Microsoft para el desarrollo de aplicaciones orientadas a la web además pone a nuestra disposición herramientas y librerías para su comprensión y estudio.

Es importante mencionar que soporta las tecnologías de programación del .Net, como lo son C# y Visual Basic, y muy pronto otras tecnologías.

Ahora empezaremos con la forma más tradicional para hacer aplicaciones Web, que es el uso de Web Forms y como editor ocuparemos Visual Studio.

1.1 | Formularios web

Creando un Sitio Web.

Para iniciar crearemos un sitio web básico, aplicando **C#** y usando **Visual Studio**, el cual ya debe estar instalado.

Creemos un sitio web usando las siguientes opciones: **Archivo // Nuevo // Proyecto...**

Como lo muestra la *Imagen 1*.

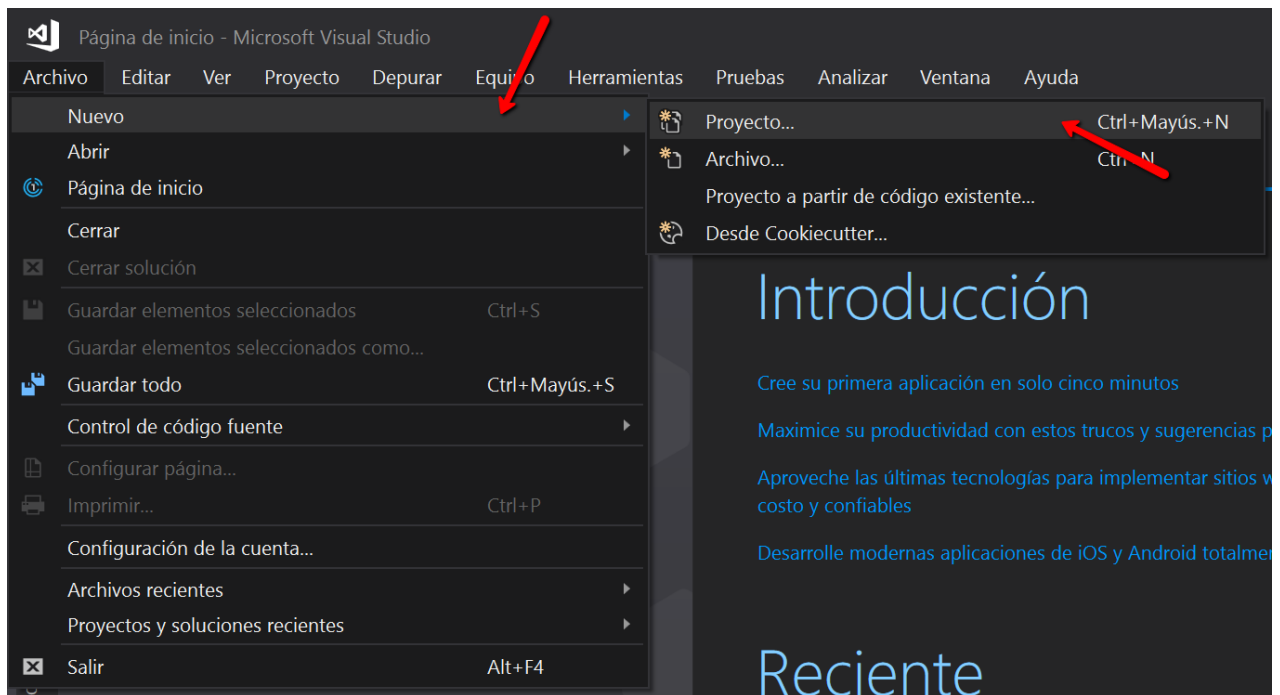


Imagen 1. Pasos para la creación de un sitio web usando C#.

Al tomar esta opción nos presentará una pantalla, como lo muestra la Imagen 2, en la cual nos permitirá escoger entre varios tipos de sitios WEB predefinidos.

Seleccionaremos la opción **“Sitio web vacío de ASP.NET”**, usamos esta opción para que nos proporcione un entorno limpio, y poder crear un sitio web desde 0, asegurándonos de usar **“C#”**, que la versión del **“.NET Framework 4.5.2”** o superior, este activa, en la parte de abajo podemos personalizar el nombre de nuestro sitio, el cual para nuestro ejemplo colocaremos el nombre de **“WebSite1”**, este es un nombre sugerido pero puedes cambiarlo por el tuyo.

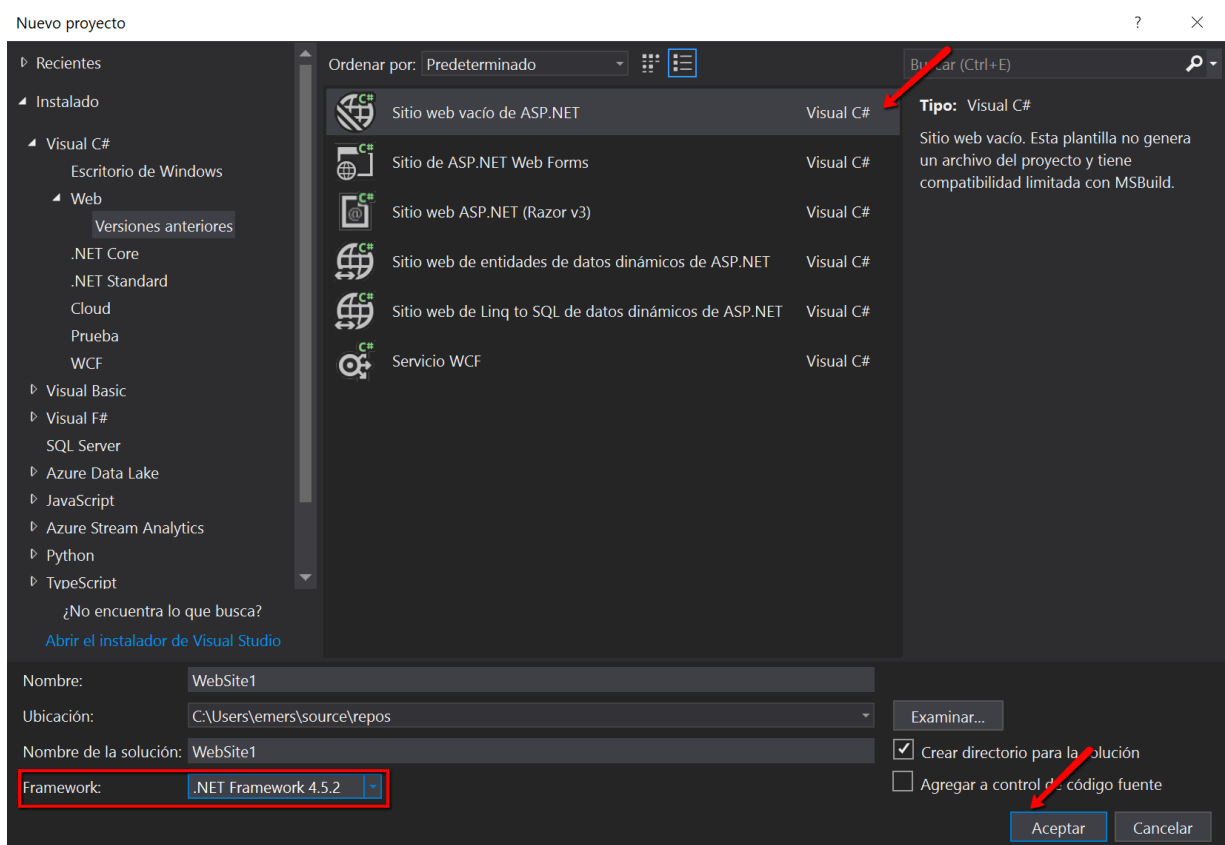


Imagen 2. Configuración de nuevo sitio en ASP.NET

Al seleccionar “**Aceptar**” se creará un sitio web vacío con el archivo: “**Web.config**”, el cual nos servirá para configurar algunos valores en nuestro sitio como permisos, conexiones a bases de datos, y que aprenderemos a configurar más adelante.

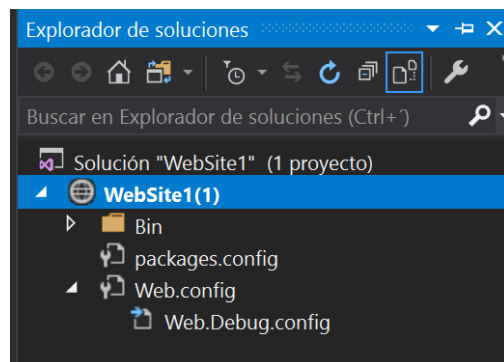


Imagen 3. Sitio básico creado en C#

Ahora para crear nuestro primer script, presionaremos clic derecho sobre el nombre del sitio, en nuestro caso “**WebSite1**”, seleccionaremos “**Agregar**”, y posteriormente “**Agregar nuevo elemento...**” según lo muestra la imagen 4.

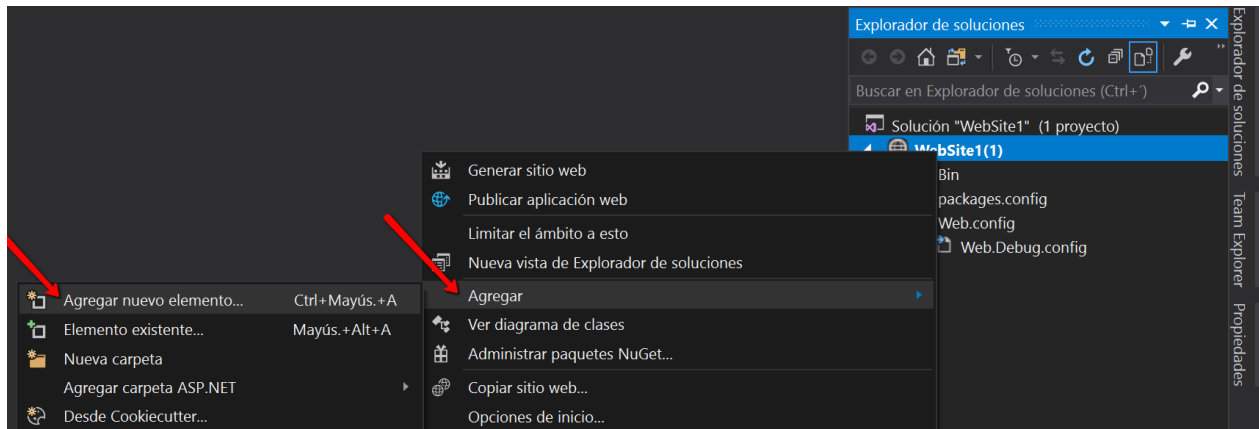


Imagen 4. Agregar un elemento nuevo.

A continuación, nos aparecerá una pantalla, que nos permitirá seleccionar, entre un gran número de plantillas, el script que deseamos usar, para nuestro ejemplo, usaremos “**Formularios Web Forms**”, según lo muestra la figura 5, asegurándonos de tener activado la opción **C#**, activado “**Poner el código en archivo independiente**”, el nombre del script para nuestro ejemplo es “**Default.aspx**”, lo ideal sería mantener dicho nombre.

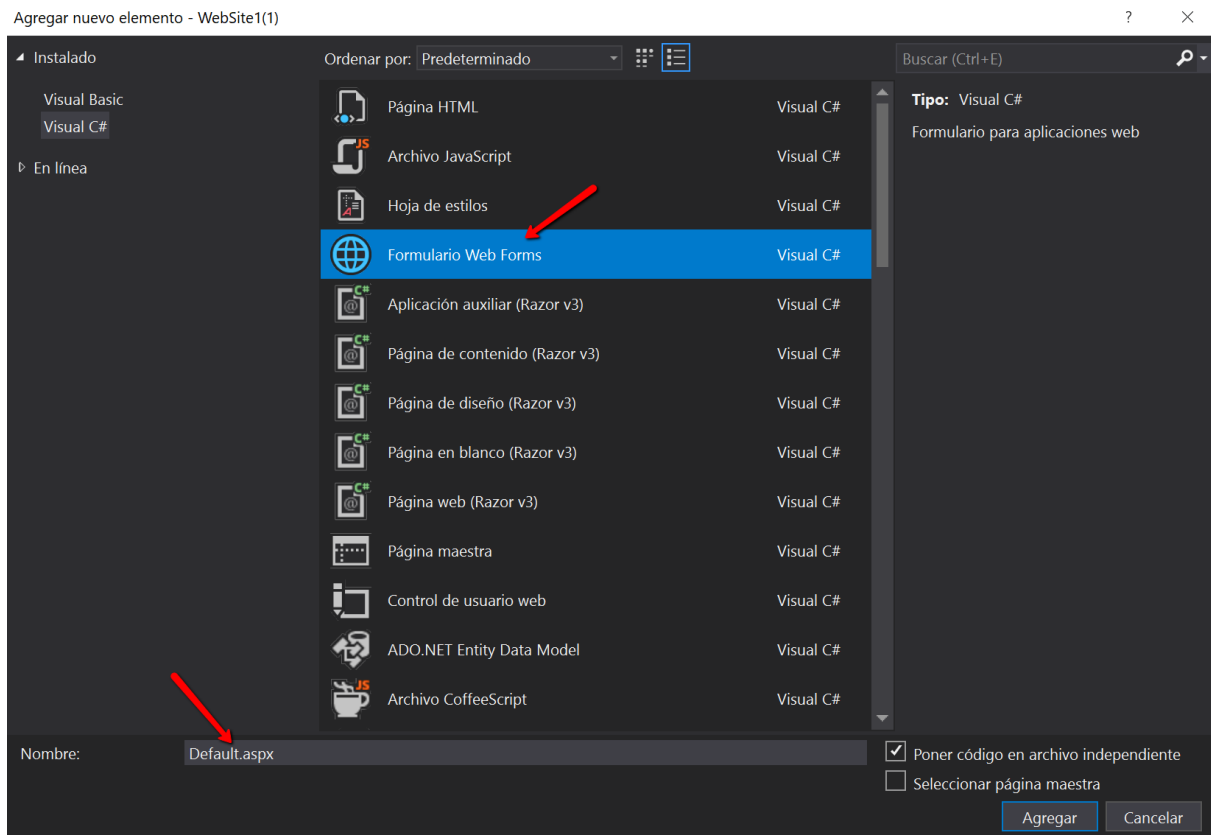


Imagen 5. Creación de script Default.aspx

Es importante mencionar que ASP.NET reconoce a la página **Default.aspx** como la principal del root (directorio principal), y esta se ejecuta automáticamente.

Una vez creado nuestro primer script nos aparecerá en el “**Explorador de soluciones**” y debajo del su respectivo “**code behind**”, con la extensión “**.cs**” como podemos ver en la figura 6, y podremos ver su estructura que es bastante parecida a la de un archivo normal de HTML, con la diferencia de la cabecera, y de la expresión **runat=“server”** lo cual indica que los componentes se compilan en el servidor, como lo muestra la Figura 7.

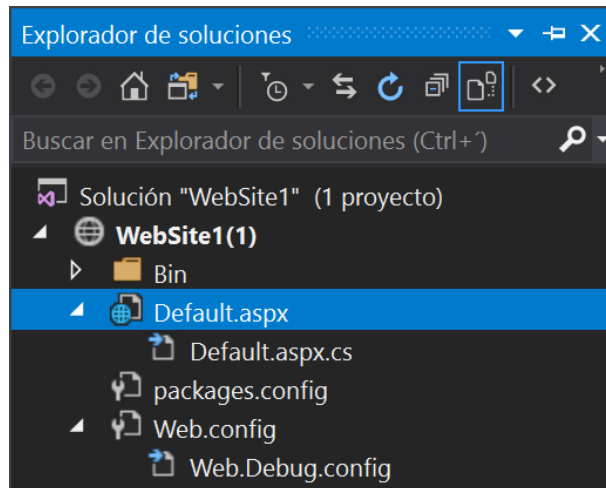


Imagen 6. Estructura de la WebSite1

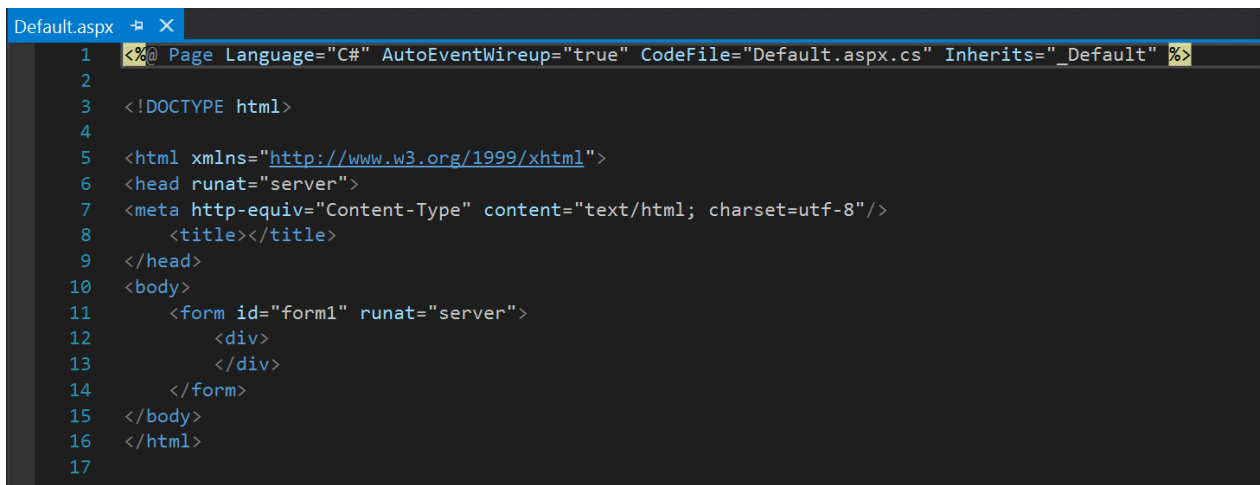


Imagen 7. Estructura de archivo Default.aspx

Es importante mencionar que el **IDE Visual Studio** tiene ya controles definidos, los cuales se pueden usar “**drag and drop**” en el modo “**diseño**”.

Ejemplo de suma de dos números.

Para este ejemplo crearemos un sitio vacío con el nombre de **ejemplo_1**, y añadiremos un script con el nombre **Default.aspx**, a continuación, utilizamos los controles que ya tiene ASP.NET y agregamos dos “**Textbox**”, un “**Button**” y por último un “**Label**”, según muestra la figura 8.

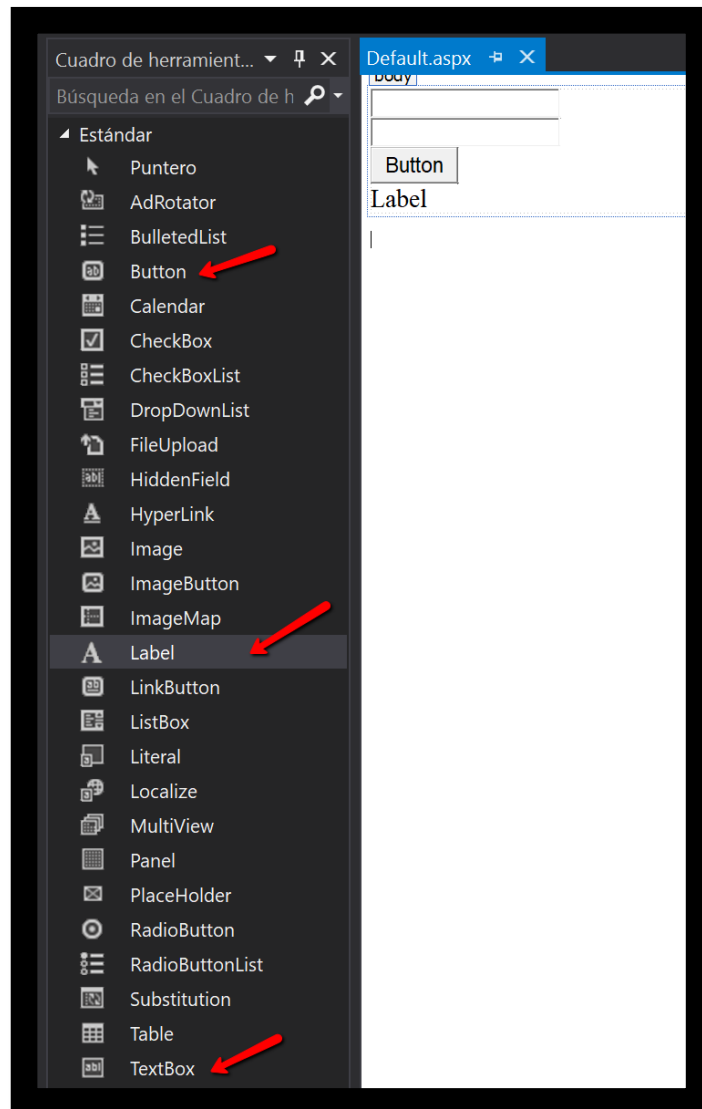


Figura 8: Ejemplo de pantalla.

Ahora para que el código no se vea sin formato, ocuparemos **CSS**, para lo cual primero crearemos un archivo que se llame “**StyleSheet.css**”, como lo muestra la figura 9.

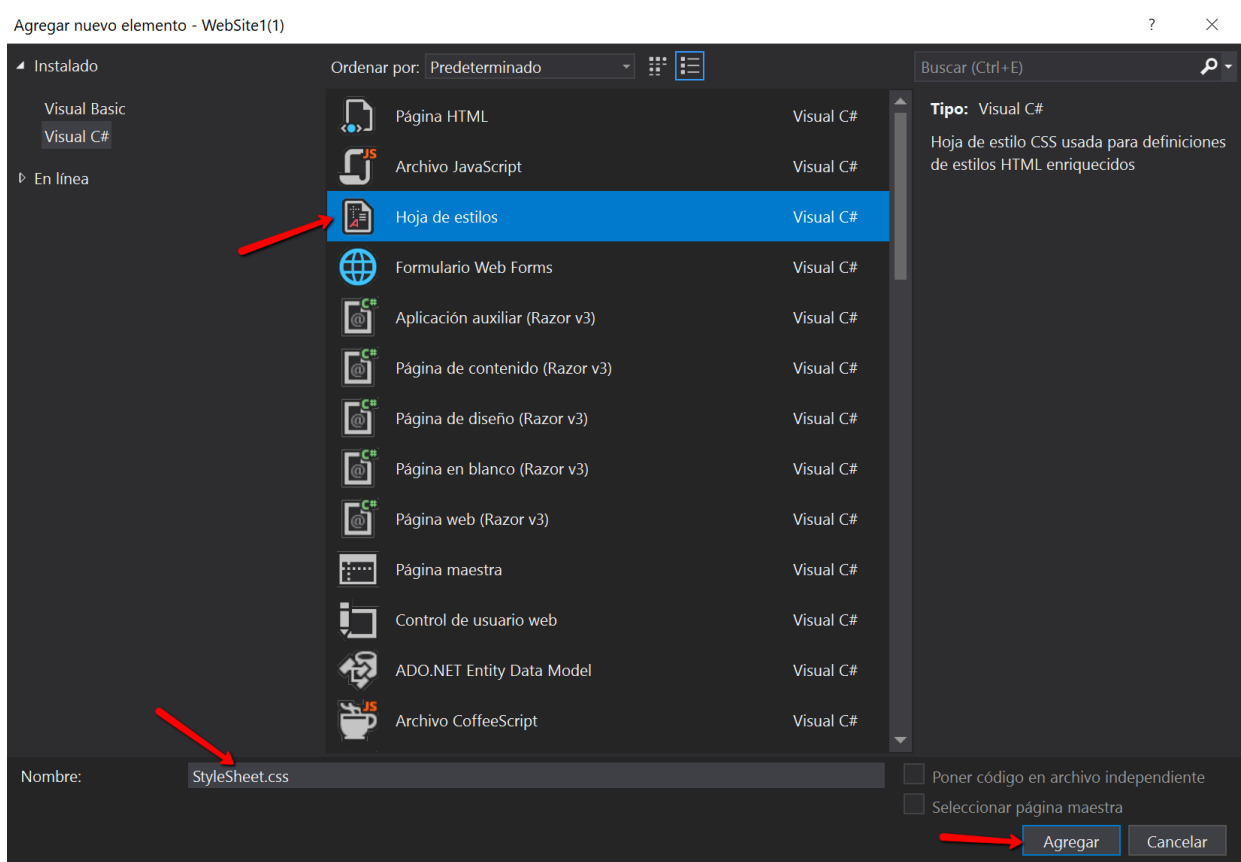


Figura 9: Como crear un archivo CSS.

Ahora configuraremos con el siguiente código el archivo forma.css

```
#caja_superior {
    width: 500px;    margin: 15px auto;    padding: 8px;    text-align: center;    background-color:
blue;
}

#caja_basica {
    width: 400px;    margin: 15px auto;    text-align: left;
padding: 8px;
}

.boton {
    padding: 15px;
margin: 15px;
}

h2 {
    color: white;
}

.resultado {
    padding: 25px;    margin: 15px;
font-size: 25px;
}
```

Código 1. Script forma.css con sus selectores.

Ahora en el script **Default.aspx** incorporaremos la configuración CSS con la siguiente línea:

```
<%@ Page Language="C#" AutoEventWireup="false" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Sumar dos numeros</title>
    <link href="StyleSheet.css" rel="stylesheet" />
  </head>
  <body>
    <div id="caja_superior">
      <h2>Suma numeros</h2>
    </div>
    <form id="form1" runat="server">
      <div id="caja_basica">
        <asp:TextBox ID="TextBox1" runat="server"
        CssClass="boton"></asp:TextBox><br />
        <asp:TextBox ID="TextBox2" runat="server"
        CssClass="boton"></asp:TextBox><br />
        <asp:Button ID="Button1" runat="server" CssClass="boton" Text="Button"
        /><br />
        <asp:Label ID="Label1" runat="server" CssClass="resultado"></asp:Label>
      </div>
    </form>
  </body>
</html>
```

Y agregamos algunos **<div>** y configuramos **CssClass** a los **Textbox**, **Button** y **Label**.

También creamos un método en el botón **“Button1”** presionando doble clic sobre él, para este ejemplo lo único que haremos es sumar simplemente dos números, sin validaciones, y usamos **“int.Parse”** para que reconozca que son números, la programación puede verse en el código 2.

```
protected void Button1_Click(object sender, EventArgs e)
{
    int valor = int.Parse(TextBox1.Text) + int.Parse(TextBox2.Text);
    Label1.Text = valor.ToString();
}
```

Código 2. Codificación del Button1.

Ahora pruebe el ejemplo y podrá sumar dos números, recuerde que no tiene validación, si coloca letras o espacios generara un error, al final el resultado puede verse en la figura.

10

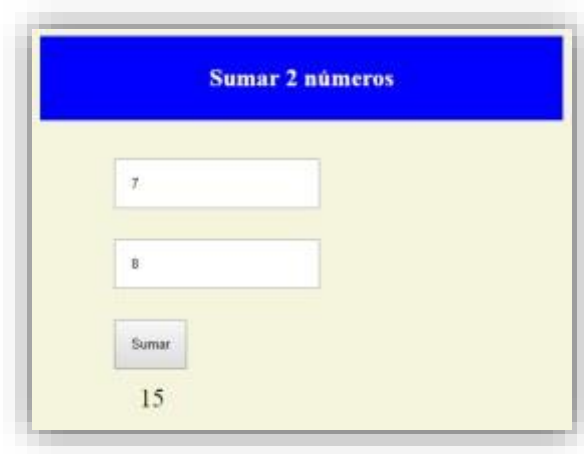
The image shows a web application window with a blue title bar that says "Sumar 2 números". Below the title bar, there are two text input fields. The first field contains the number "7" and the second field contains the number "8". Below these fields is a button labeled "Sumar". At the bottom of the application area, the result "15" is displayed.

Figura 10. Vista de la Aplicación

Ejercicios:

1. Cree una calculadora que sume, reste, multiplique o divida dos números, si el divisor es 0 que no haga la operación.
2. Cree un formulario WEB donde un estudiante pueda colocar sus notas, tres en total y que puede obtener un promedio, si la nota es 10 que lo felicite, si es menor que 10 pero 7 o mayor, que le coloque aprobado, si es menor que 7 coloque aplazado y a los de 4 para abajo, sugerir que visite al tutor.
3. Crear un ejemplo usando otros controles WEB, como por ejemplo, **listbox**, **Radiobutton**, **Drowdownlist**.
4. Elaborar un formulario de registro que me pida los datos completos de una persona, nombres, apellidos, teléfono, ciudad de nacimiento, correo, peso, sexo y estatura, usando varios controles web.
5. Cree un script que alimente a un **listbox**, a partir de un formulario de ASP.NET.

Información importante.

Es importante tomar en cuenta con el framework que utilizamos para crear las aplicaciones, ya que, si queremos implementarlas en un servidor comercial, debemos de asegurarnos que soporte dichas versiones.

También es importante entender la diferencia entre una aplicación de escritorio escrita en C# y una web, ya que tienen algunas funcionalidades diferentes.

Y por último el uso del page load puede brindarnos muchas ventajas de programación.

1.2 | Controles de Validación de ASP.NET

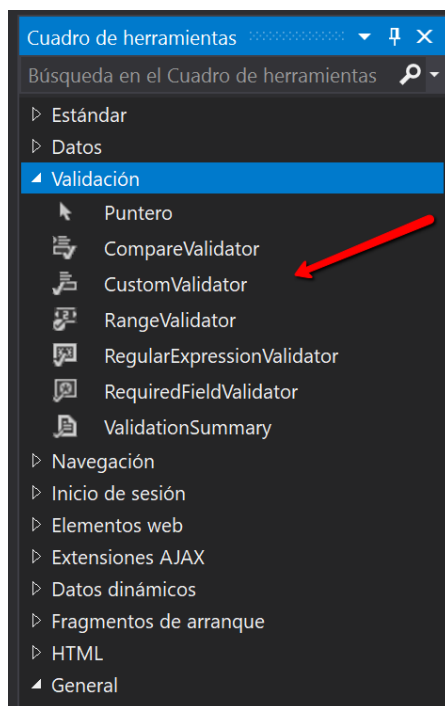


Figura 11. Controles de validación de asp.net

Los controles de servidor de validación son un conjunto de controles que permiten validar un control de servidor de entrada asociado, como un control **TextBox**, y mostrar un mensaje personalizado cuando se produce un error en la validación. Cada control de validación realiza un tipo determinado de validación. Por ejemplo, se puede validar frente a un valor específico o un intervalo de valores mediante los controles **CompareValidator** y **RangeValidator**, respectivamente. Se puede incluso definir criterios de validación propios mediante el control **CustomValidator**. Dado que el mensaje de error se muestra en el control de validación, se puede controlar el lugar donde se muestra el mensaje en la página Web colocando el control de validación en el lugar deseado. También se puede mostrar un resumen de los resultados de todos los controles de validación en la página mediante el control **ValidationSummary**.

De manera predeterminada, la validación se realiza al hacer clic en un control de botón, como **Button**, **ImageButton** o **LinkButton**. Para impedir que se realice la validación al hacer clic en un control de botón, establezca la propiedad **CausesValidation** del control de botón en false. Esta propiedad suele estar establecida en false para un botón

cancel o **clear** con el fin de impedir que se realice la validación cuando se hace clic en el botón. Propiedades básicas de los controles de Validación

La siguiente tabla muestra las principales propiedades que se aplican a todos los controles de validación.

Propiedad	Descripción
ControlToValidate	Id. de programación del control de salida que evaluará el control de validación. Si no es un Id. Legítimo, se iniciará una excepción.
Display	<p>Modo en que se muestra el control de validación especificado. Esta propiedad puede ser uno de los siguientes valores:</p> <p>Ninguno — El control de validación jamás se muestra en línea. Utilice esta opción cuando desee mostrar el mensaje de error sólo en un control ValidationSummary</p> <p>Estático — El control de validación muestra un mensaje de error si se produce un error en la validación. Se asigna un espacio en la página Web para el mensaje de error incluso si el control de entrada supera la validación. No cambia el diseño de la página cuando el control de validación muestra su mensaje de error. Como el diseño de página es estático, si hay varios controles de validación para el mismo control de entrada, éstos deberán ocupar distintas ubicaciones en la página.</p> <p>Dinámico — El control de validación muestra un mensaje de error si se produce un error en la validación. El espacio para el mensaje de error se asigna dinámicamente en la página cuando se produce un error en la validación. De este modo, varios controles de validación pueden compartir la misma ubicación física en la página.</p>
EnableClientScript	Indica si está habilitada la validación en el cliente. Para deshabilitar la validación en el cliente en los exploradores que admitan esta función, establezca la propiedad EnableClientScript en false.
Enabled	Indica si está habilitado el control de validación. Para impedir que el control de validación valide un control de entrada, establezca esta propiedad en false.
ErrorMessage	Mensaje de error que se va a mostrar en el control ValidationSummary si se produce un error en la validación. Si no está establecida la propiedad Text del control de validación, también se muestra este texto en el control de validación cuando se produce un error en la validación. Se utiliza normalmente la propiedad ErrorMessage para proporcionar diferentes mensajes para el control de validación y el control ValidationSummary.
ForeColor	Especifica el color en el que se va a mostrar el mensaje en línea cuando se produce un error en la validación.

IsValid	Indica si el control de entrada especificado por la propiedad ControlToValidate se determina como válido.
Text	Si está establecida esta propiedad, se muestra este mensaje en el control de validación cuando se produce un error en la validación. Si no está establecida esta propiedad, el texto especificado en la propiedad ErrorMessage se muestra en el control.

Descripción de los controles de Validación

RequiredFieldValidator

Utilice el control **RequiredFieldValidator** para convertir un control de entrada en un campo obligatorio. El control de entrada no supera la validación si el valor que contiene no cambia con respecto al valor inicial cuando se realizó la validación. Esto impide que el usuario deje el control de entrada asociado sin modificar. De manera predeterminada, el valor inicial es una cadena vacía (""), que indica que se ha de especificar un valor en el control de entrada para que éste supere la validación.

A veces, se desea tener un valor inicial que no sea una cadena vacía. Esto resulta útil cuando hay un valor predeterminado para un control de entrada y se desea que el usuario seleccione otro valor. Por ejemplo, puede haber un control **ListBox** con una entrada seleccionada, de manera predeterminada, que contiene instrucciones para que el usuario seleccione un elemento de la lista. El usuario debe seleccionar un elemento del control, pero se desea que el usuario no seleccione el elemento que contiene las instrucciones. Para impedir que el usuario seleccione este elemento, especifique su valor como valor inicial. Si el usuario selecciona este elemento, **RequiredFieldValidator** mostrará su mensaje de error. Para especificar el valor inicial del control de entrada asociado, establezca la propiedad **InitialValue**.

Nota: La propiedad **InitialValue** no establece el valor predeterminado del control de entrada. La propiedad **InitialValue** no tiene que coincidir ni siquiera con el valor predeterminado del control de entrada. Indica simplemente el valor que el usuario no podrá especificar en el control de entrada. El control de entrada no superará la validación si contiene este valor al realizarse la validación.

Se pueden asociar varios controles de validación al mismo control de entrada. Se puede utilizar, por ejemplo, un control de validación **RequiredFieldValidator** para asegurar que se han escrito datos en un control y usar al mismo tiempo un control de validación **RangeValidator** para comprobar que estos datos están comprendidos en un intervalo especificado.

CompareValidator

El control **CompareValidator** permite comparar el valor especificado por el usuario en un control de entrada, como un control **TextBox**, con el valor especificado en otro control de entrada o con un valor constante. También se puede usar el control **CompareValidator** para determinar si el valor especificado en un control de entrada se puede convertir al tipo de datos especificado por la propiedad **Type**.

Especifique el control de entrada que desee validar estableciendo la propiedad **ControlToValidate**. Si desea comparar un control de entrada específico con otro control de entrada, establezca la propiedad **ControlToCompare** en el nombre del control que desee comparar.

En lugar de comparar el valor de un control de entrada con otro control de entrada, se puede comparar el valor de un control de entrada con un valor constante. Especifique el valor constante con el que desee comparar estableciendo la propiedad **ValueToCompare**.

La propiedad **Operator** permite especificar el tipo de comparación que se va a realizar, como mayor que, igual que, etc. Si se establece la propiedad **Operator** en **ValidationCompareOperator**. **DataTypeCheck**, el control **CompareValidator** omitirá las propiedades **ControlToCompare** y **ValueToCompare** e indicará simplemente si el valor especificado en el control de entrada puede convertirse al tipo de datos especificado por la propiedad **Type**.

Nota Si el control de entrada está vacío, no se llamará a ninguna función de validación y la validación se realizará correctamente. Utilice un control **RequiredFieldValidator** para impedir que el usuario omita un control de entrada.

RangeValidator

El control RangeValidator permite comprobar si la entrada de un usuario se encuentra entre un límite inferior y un límite superior especificados. Se pueden comprobar los intervalos entre pares de números, caracteres alfabéticos y fechas. Los límites se expresan como constantes.

Utilice la propiedad **ControlToValidate** para especificar el control de entrada que se va a validar. Las propiedades **MinimumValue** y **MaximumValue** especifican los valores mínimo y máximo del intervalo válido, respectivamente.

La propiedad **Type** se utiliza para especificar el tipo de datos de los valores que se van a comparar. Los valores que se van a comparar se convierten a este tipo de datos antes de realizarse cualquier comparación.

Nota: Si el control de entrada está vacío, no se llamará a ninguna función de validación y la validación se realizará correctamente. Utilice un control **RequiredFieldValidator** para impedir que el usuario omita un control de entrada. De manera similar, si el valor del control de entrada no se puede convertir al tipo de datos especificado por la propiedad **Type**, la validación también se llevará a cabo correctamente. Se recomienda encarecidamente utilizar un control **CompareValidator** adicional, con su propiedad **Operator** establecida en **ValidationCompareOperator.DataTypeCheck**, con el fin de comprobar el tipo de datos del valor de entrada.

Nota El control **RangeValidator** iniciará una excepción si el valor especificado por la propiedad **MaximumValue** o **MinimumValue** no se puede convertir al tipo de datos especificado por la propiedad **Type**.

RegularExpressionValidator

El control **RegularExpressionValidator** se utiliza para determinar si el valor de un control de entrada coincide con un modelo definido por una expresión regular. Este tipo de validación permite comprobar secuencias de caracteres previsibles, como las de los números de la seguridad social, las direcciones de correo electrónico, los números de teléfono y los códigos postales, entre otras.

Nota Si el control de entrada está vacío, no se llamará a ninguna función de validación y la validación se realizará correctamente. Utilice un control **RequiredFieldValidator** para impedir que el usuario omita un control de entrada.

La validación se realiza tanto en el cliente como en el servidor, salvo en el caso de que el explorador no admita la validación en el cliente o ésta haya sido explícitamente deshabilitada (estableciendo la propiedad `EnableClientScript` en `false`).

Utilice la propiedad **ValidationExpression** para especificar la expresión regular utilizada para validar el control de entrada. La sintaxis de la validación mediante expresiones regulares presenta algunas diferencias en el cliente y en el servidor. En el cliente, se utiliza la sintaxis de expresiones regulares de **JScript**. En el servidor, se utiliza la sintaxis **Regex**. Dado que la sintaxis de las expresiones regulares de **JScript** es un subconjunto de la sintaxis **Regex**, se recomienda utilizar la primera para obtener los mismos resultados en el cliente y en el **servidor.validator**

ValidationSummary

El control **ValidationSummary** permite resumir los mensajes de error de todos los controles de validación de una página Web en una sola ubicación. El resumen puede aparecer en forma de lista, lista con viñetas o un único párrafo, en función del valor de la propiedad **DisplayMode**. El mensaje de error que se muestra en el control **ValidationSummary** para cada control de validación en la página viene especificado por la propiedad **ErrorMessage** de cada control de validación. Si no está establecida la propiedad **ErrorMessage** del control de validación, no se mostrará ningún mensaje de error en el control **ValidationSummary** correspondiente a dicho control de validación.

También se puede especificar un título personalizado en la sección de encabezado del control **ValidationSummary** estableciendo la propiedad **HeaderText**.

Para controlar si se muestra o se oculta el control **ValidationSummary**, establezca la propiedad **ShowSummary**. El resumen también se puede mostrar en un cuadro de mensaje estableciendo la propiedad **ShowMessageBox** en true.

Ejercicios:

1. Utilizando el ejercicio de calculadora hecho en el punto anterior, agregar los controles de validación que considere necesario para que cada una de las operaciones se realicen exitosamente.