 <b>Instituto Superior Santo Domingo</b>	<b><u>UNIDAD Nº :3</u></b>  <b>Recursos de ASP.Net.</b>	<b><u>TEMAS:</u></b>  <b>Controles de validación.</b>	<b>Clase</b>  <b>7</b>
------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------	-------------------------------------------------------------	------------------------------

### Objetivos:

- Identificar los controles de validación propuestos en ASP.Net.
- Elegir el más adecuado para cada problema.
- Crear algoritmos de validación en el servidor.

### Introducción

Hay seis controles Web para la validación de datos de entrada que se pueden incorporar en un Formulario Web.

*RequiredFieldValidator:* Facilita la validación de un dato del formulario chequeando que el mismo tenga algún valor.

*RangeValidator:* Facilita la validación de un dato del formulario contra un valor mínimo y máximo.

*CompareValidator:* Facilita la validación de un dato del formulario contra un valor fijo u otro campo del formulario.

*CustomValidator:* Facilita la validación de un dato del formulario usando una subrutina propia.

*RegularExpressionValidator:* Facilita la validación de un dato del formulario contra una expresión.

*ValidationSummary:* Agrupa los mensajes de error de otros controles en una parte de la página.

Todos los controles de validación tienen tres propiedades fundamentales: ControlToValidate, Text y IsValid. Todos los controles derivan de la clase BaseValidator.

La propiedad ControlToValidate contiene la referencia del control del formulario que queremos validar.

La propiedad Text almacena el mensaje de error que queremos que se muestre en la página.

Por último la propiedad IsValid almacena True en caso que el control pase el test de validación.

Quando empleamos controles de validación con el Explorer 4.0 o superior, los controles automáticamente usan funciones en JavaScript en el cliente. Esto significa que los controles pueden inmediatamente mostrar los

mensajes de error en el browser mientras el usuario está completando el formulario. Si hay algún error en la página el código JavaScript previene que el usuario pueda enviar los datos al servidor.

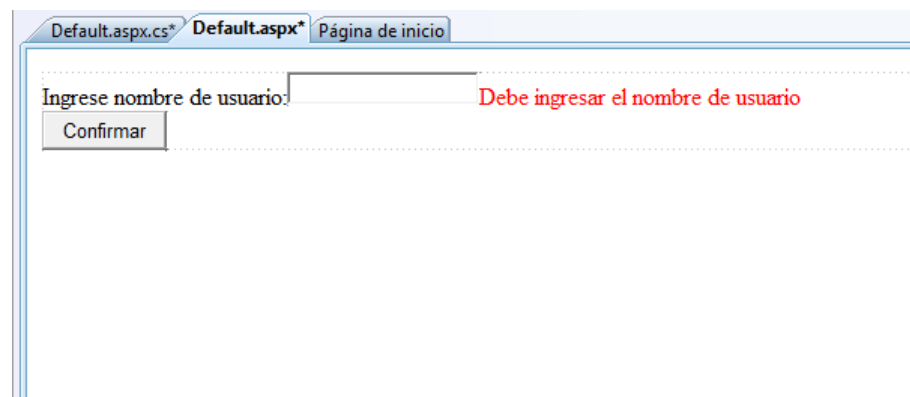
En caso de emplear navegadores más antiguos los controles que veremos seguirán funcionando, pero la validación se realizará en el servidor.

## Controles de validación

### ***Control: RequiredFieldValidator***

Para probar este control haremos una página que solicite el nombre de usuario (mostraremos un error si el operador no ingresa texto en el TextBox).

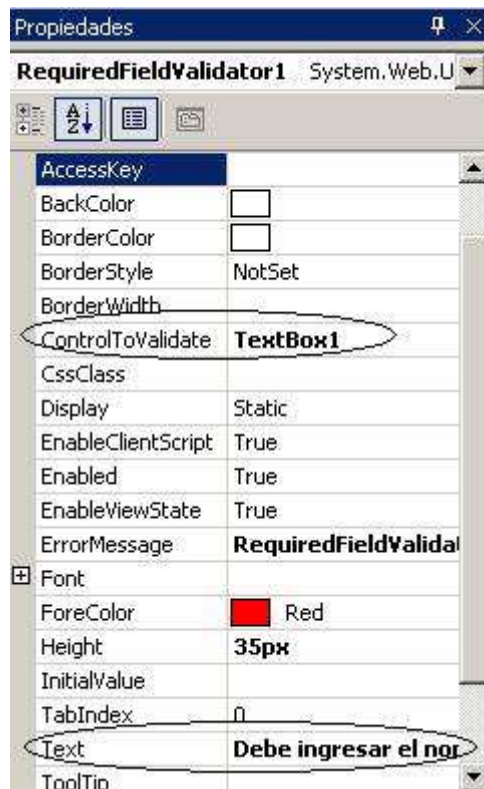
La interface visual es la siguiente:

The screenshot shows a web browser window with three tabs: 'Default.aspx.cs\*', 'Default.aspx\*', and 'Página de inicio'. The main content area contains a form with the label 'Ingrese nombre de usuario:' followed by a text input field. Below the input field is a 'Confirmar' button. To the right of the input field, a red error message reads 'Debe ingresar el nombre de usuario'.

El mensaje en rojo debe aparecer si presionamos el botón “Confirmar” y no se ingresó texto en el TextBox.

En el Visual Studio .Net confeccionamos el formulario web disponiendo uno control de tipo TextBox, un Button y un RequiredFieldValidator que se encuentra en la pestaña “Validación” del “Cuadro de herramientas”.

El control RequiredFieldValidator es importante inicializar las siguientes propiedades:



Cuando ejecutemos la página podemos ver el código que llega al navegador (en ella veremos las funciones en JavaScript que automáticamente el ASP.NET nos crea para facilitar la validación).

El código HTML completo de la página es el siguiente:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!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>

      Ingrese nombre de usuario:<asp:TextBox
ID="TextBox1" runat="server"></asp:TextBox>
      <asp:RequiredFieldValidator
ID="RequiredFieldValidator1" runat="server"
ControlToValidate="TextBox1"
ErrorMessage="RequiredFieldValidator">Debe ingresar el
nombre de usuario</asp:RequiredFieldValidator>

    </div>
```

```

        <asp:Button ID="Button1" runat="server"
onclick="Button1_Click"
        Text="Confirmar" />
    </form>
</body>
</html>

```

Como sabemos este código HTML se genera en forma automática cuando creamos cada control y configuramos sus propiedades.

Luego si queremos que al presionar el botón se redirija a otra página en caso de haber ingresado un nombre de usuario debemos codificar el método Clic para dicho botón:

```

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

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender,
EventArgs e)
    {
        if (this.IsValid)
        {
            this.Response.Redirect("Default2.aspx");
        }
    }
}

```

La propiedad IsValid del WebForm almacena true si todos los controles de validación dispuestos en el formulario se validan correctamente. Es decir en este problema si se ingresa algún carácter en el control TextBox luego se puede pasar a la página "Default2.aspx".

## Control: RangeValidator

El control RangeValidator especifica un valor mínimo y máximo para un control TextBox. Podemos utilizar el control para chequear el rango de enteros, fechas, cadenas o valores reales.

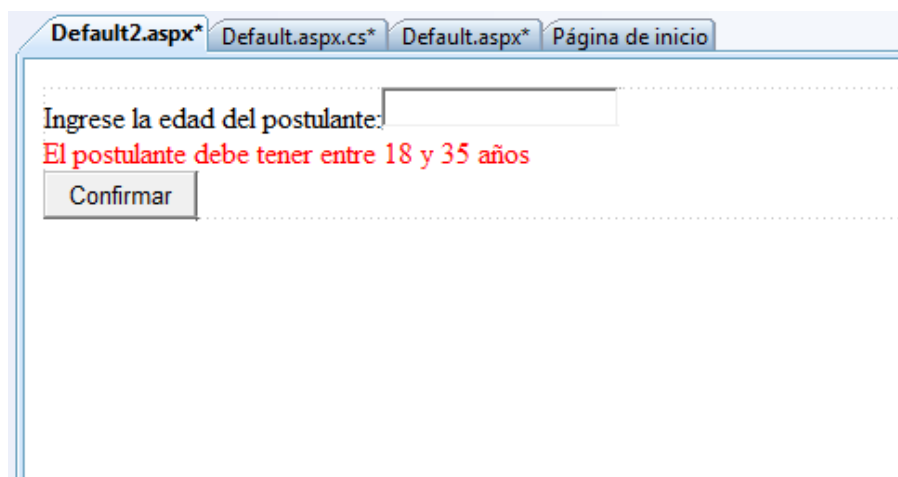
Las propiedades más importantes del control son:

- ControlToValidate El campo del formulario a validar.
- MinimumValue El valor mínimo a validar en el rango de valores.
- MaximumValue El valor máximo a validar en el rango de valores.
- Text El mensaje de error a mostrar.
- Type El tipo de comparación a ejecutar (valores posibles: String, Integer, Double, Date, Currency).

Para probar este control haremos una página que solicite ingresar la edad de una persona que se postula para un trabajo (la misma debe estar en el rango de 18 a 35 años).

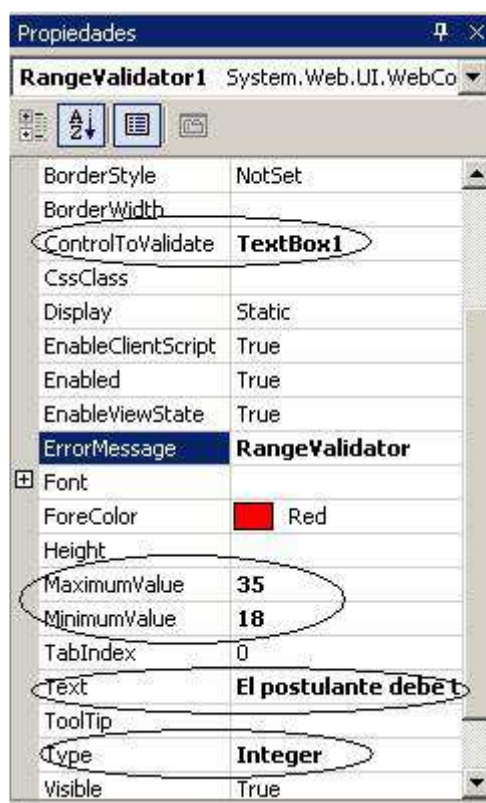
Disponemos sobre el formulario los siguientes objetos: Label, TextBox, Button y un RangeValidator.

La interface que debemos implementar es la siguiente:



The screenshot shows a web browser window with a single tab titled 'Default2.aspx\*'. The page content includes a label 'Ingrese la edad del postulante:' followed by a text input field. Below the input field, a red error message is displayed: 'El postulante debe tener entre 18 y 35 años'. At the bottom of the form is a button labeled 'Confirmar'.

El objeto RangeValidator lo debemos configurar con los siguientes valores:



The screenshot shows the 'Propiedades' (Properties) window for a 'RangeValidator1' control. The control is of type 'System.Web.UI.WebCo'. The following properties are visible and configured:

Property	Value
BorderStyle	NotSet
BorderWidth	
ControlToValidate	TextBox1
CssClass	
Display	Static
EnableClientScript	True
Enabled	True
EnableViewState	True
ErrorMessage	RangeValidator
Font	
ForeColor	Red
Height	
MaximumValue	35
MinimumValue	18
TabIndex	0
Text	El postulante debe t
ToolTip	
Type	Integer
Visible	True

Si ejecutamos la página veremos que el mensaje aparece si ingresamos una edad que esté fuera del rango de 18 a 35 años.

El código a implementar al presionar el botón confirmar es el siguiente:

```

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

public partial class Default2 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender,
    EventArgs e)
    {
        if (this.IsValid)
        {
            this.Response.Redirect("Default3.aspx");
        }
    }
}

```

Es decir redireccionamos a la próxima página en caso que todos los controles de validación del formulario se verifiquen correctos (en este problema solo tenemos un control de tipo RangeValidator, pero en muchos casos veremos que en un formulario puede haber más de un control de validación)

Si quisiéramos solo validar un control determinado del WebForm la condición sería:

```

if (this.RangeValidator1.IsValid)
{
    this.Response.Redirect("Default3.aspx");
}

```

Es decir verificamos la propiedad IsValid del control RangeValidator (si tenemos un solo control en el formulario preguntar por la propiedad IsValid del webform o del RangeValidator el resultado será idéntico).

## ***Control: CompareValidator***

El control CompareValidator permite comparar un valor de un control con otro control o comparar el valor de un control con un valor fijo. Las propiedades más importantes son:

- **ControlToValidate** El campo del formulario a validar.
- **ControlToCompare** El campo del formulario contra el cual se efectúa la comparación.
- **Operator** El operador a utilizarse en la comparación (los valores posibles son Equal, NotEqual, GreaterThan,

GreaterThanEqual, LessThan, LessThanEqual y DataTypeCheck).

- Text El mensaje de error a mostrar.
- Type El tipo de comparación a ejecutar (valores posibles String, Integer, Double, Date, Currency).
- ValueToCompare El valor fijo a comparar.

Para probar este control implementaremos una página que realizaría el alta de la tabla usuarios (debe permitir el ingreso del nombre de usuario y su clave, esta última dos veces, con el objetivo de asegurarse que la ingresó correctamente), emplearemos un objeto de la clase CompareValidator para validar el ingreso repetido de la clave.

La interface visual de la página es:

Nombre de usuario:

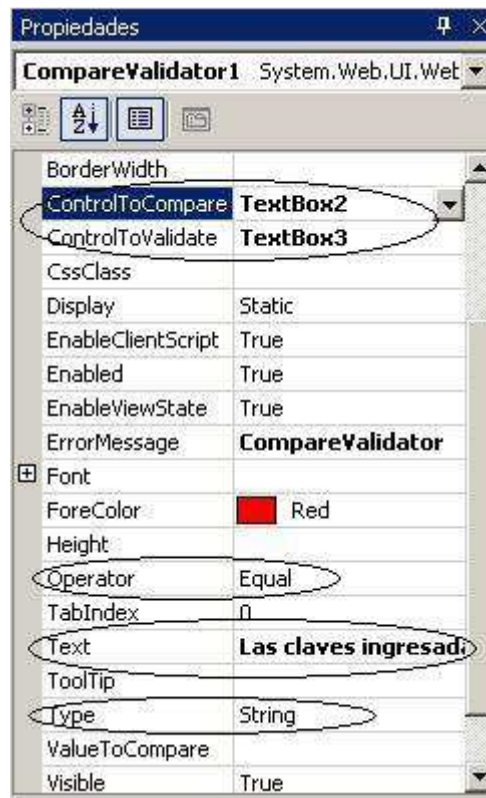
Clave:

Repita clave:

Las claves ingresadas son distintas

Confirmar

Al objeto CompareValidator le configuramos las propiedades de la siguiente manera:



Es importante inicializar la propiedad ControlToValidate con el objeto TextBox que carga la segunda clave, luego que el operador carga la clave se procede a validar si el texto ingresado coincide en el TextBox que hemos inicializado la propiedad ControlToCompare.

El código a implementar al presionar el botón “Confirmar”:

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

public partial class Default3 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected void Button1_Click(object sender,
    EventArgs e)
    {
        if (this.IsValid)
        {
            this.Response.Redirect("Default5.aspx");
        }
    }
}
```

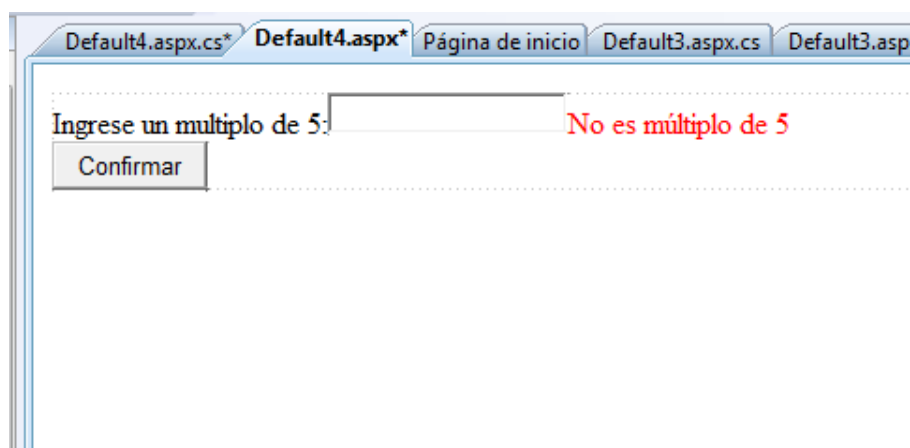
## **Control: CustomValidator**



El control CustomValidator permite validar el campo de un formulario con una función de validación propia. Debemos asociar nuestro control CustomValidator con un evento propio.

Para probar este control implementaremos una página que solicite el ingreso de un número múltiplo de 5, en caso de ingresar un valor incorrecto mostraremos un mensaje de error.

La interface a implementar es la siguiente:



Primero configuramos las propiedades del objeto CustomValidator con:



Ahora debemos codificar el evento `ServerValidate` que tiene el objeto `CustomValidator1`, a esto lo hacemos desde la ventana de código y lo podemos generar automáticamente haciendo doble clic sobre el control en la ventana de diseño:

```
protected void
CustomValidator1_ServerValidate(object source,
ServerValidateEventArgs args)
{
    int valor;
    valor=int.Parse(this.TextBox1.Text);
    if (valor % 5 == 0)
        args.IsValid = true;
    else
        args.IsValid = false;
}
```

El parámetro `args` tiene una propiedad fundamental llamada `IsValid` que debemos asignarle el resultado de nuestra validación. En nuestro ejemplo almacenamos el número ingresado en la variable `valor`, luego mediante el operador `%` (resto de una división) verificamos si es cero, en caso afirmativo inicializamos la propiedad `IsValid` del objeto `args` con el valor `True`, en caso contrario, es decir que el número ingresado no sea un múltiplo de 5 almacenamos el valor `False`.

Cuando se presiona el botón confirmar tenemos:

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (this.IsValid)
    {
        this.Response.Redirect("Default5.aspx");
    }
}
```

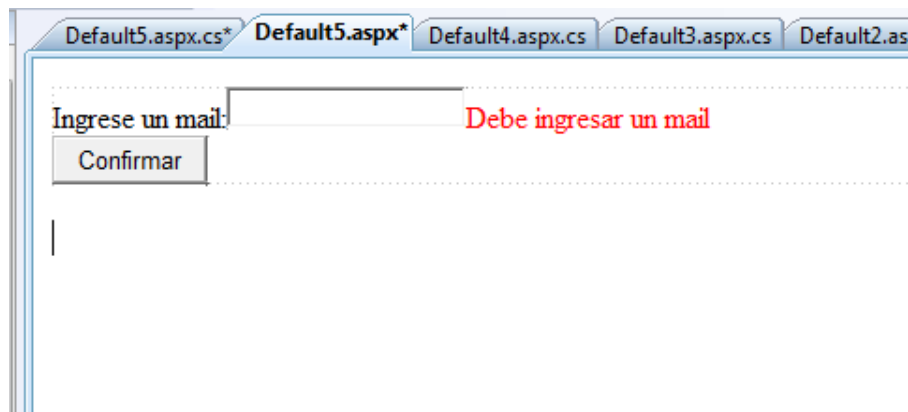
## ***Control: RegularExpressionValidator***

El control `RegularExpressionValidator` permite validar el valor de un campo de un formulario con un patrón específico, por ejemplo un código postal, un número telefónico, una dirección de mail, una URL etc.

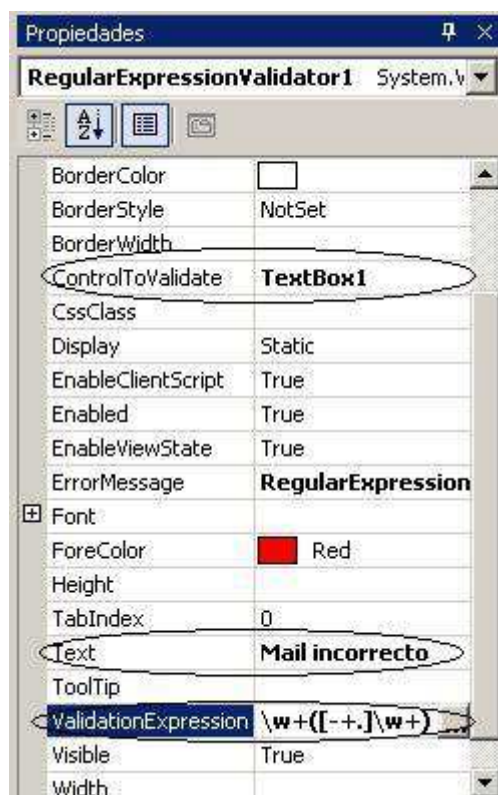
El planteo de una `RegularExpression` es bastante compleja, pero afortunadamente el Visual Studio .Net provee una serie de expresiones regulares preconfiguradas y otras las podemos buscar en internet.

Para probar este control, haremos una página que solicite el ingreso de un mail y mostraremos un error en caso que el usuario ingrese un mail mal formado.

La interface visual de la página es la siguiente:



Al objeto de la clase `RegularExpressionValidator` le configuramos las propiedades con los siguientes valores:



Si ejecutamos el programa podremos ver que al abandonar el foco del `TextBox` aparecerá el mensaje de error en caso de ingresar un mail incorrecto:

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

public partial class Default5 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
```

```

    {
    }
    protected void Button1_Click(object sender,
EventArgs e)
    {
        if (this.IsValid)
        {
            this.Response.Redirect("Default6.aspx");
        }
    }
}

```

## Control: ValidationSummary

Cuando tenemos formularios con gran cantidad de controles puede llegar a ser dificultoso ubicar los errores en la página. El Framework de la .Net trae otra clase llamada ValidationSummary que muestra todos los errores de la página agrupados en una parte de pantalla. Para utilizar el control ValidationSummary es necesario fijar el valor de la propiedad ErrorMessage para cada control de validación que tiene la página. Es importante no confundir la propiedad Text que aparece en la misma posición donde la disponemos con la propiedad ErrorMessage que contiene el mensaje de error que mostrará el control ValidationSummary.

Para probar este control haremos una página que solicite la carga del nombre de usuario y su clave en forma obligatoria (mostrando un mensaje de error en caso de dejar vacío los TextBox).

La interface de la página es la siguiente:

The screenshot shows a web application window with several tabs at the top: 'Default6.aspx\*', 'Default5.aspx.cs\*', 'Default5.aspx\*', and 'Default4.aspx'. The main content area contains a form with two text input fields. The first field is labeled 'Usuario:' and the second is labeled 'Clave:'. Both fields have a red asterisk to their right, indicating they are required. Below these fields is a button labeled 'Button'. At the bottom of the form, there is a red bulleted list with two items: 'Mensaje de error 1.' and 'Mensaje de error 2.'

Debemos disponer los siguientes objetos sobre el formulario:

1Button, 2 TextBox, 2 RequiredValidator y un objeto de la clase ValidationSummary.

La propiedad text de los objetos RequiredValidator las inicializamos con un (\*) asterisco y las propiedades ErrorMessage con las cadenas: "Debe ingresar el nombre de usuario" y "Debe ingresar la clave" respectivamente. En cuanto al objeto de la clase ValidationSummary no debemos hacer ninguna configuración específica, solo ubicarlo dentro de la página.