# CLASE 8

Detallamos los ejercicios de este laboratorio, en esta ocasión un ejercicio guiado paso a paso con el propósito de conocer el uso de los controles Validators para realizar validaciones de datos en el cliente, y ejercicio de acceso a datos.

* Ejercicio 1: Uso de controles Validators
* Ejercicio 2: Acceso a datos simple con aplicación de consola
* Ejercicio 3: Aplicación con CRUD en ADO.Net y WebForms
* Ejercicio 4: Ver capa acceso a datos de Ejercicio Integrador

## Ejercicio 1: Uso de controles Validators

### 

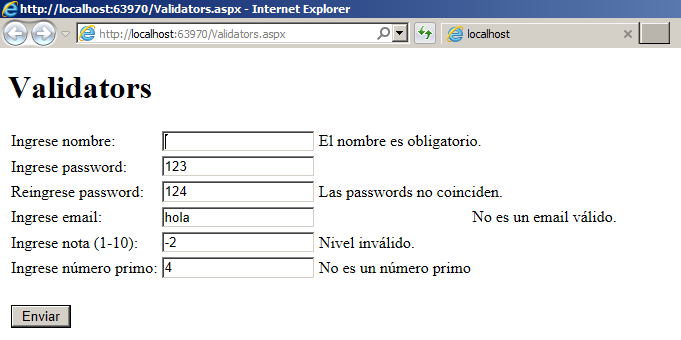
### Introducción

En la sección de descargas de esta clase, ubique el proyecto **Validators**. Bájelo y cárguelo a Visual Studio.

### 

### Objetivos

Ejercitar con los distintos controles **Validators**, distintas situaciones de validaciones que datos en el cliente, muy comunes en formularios/páginas de carga de datos.

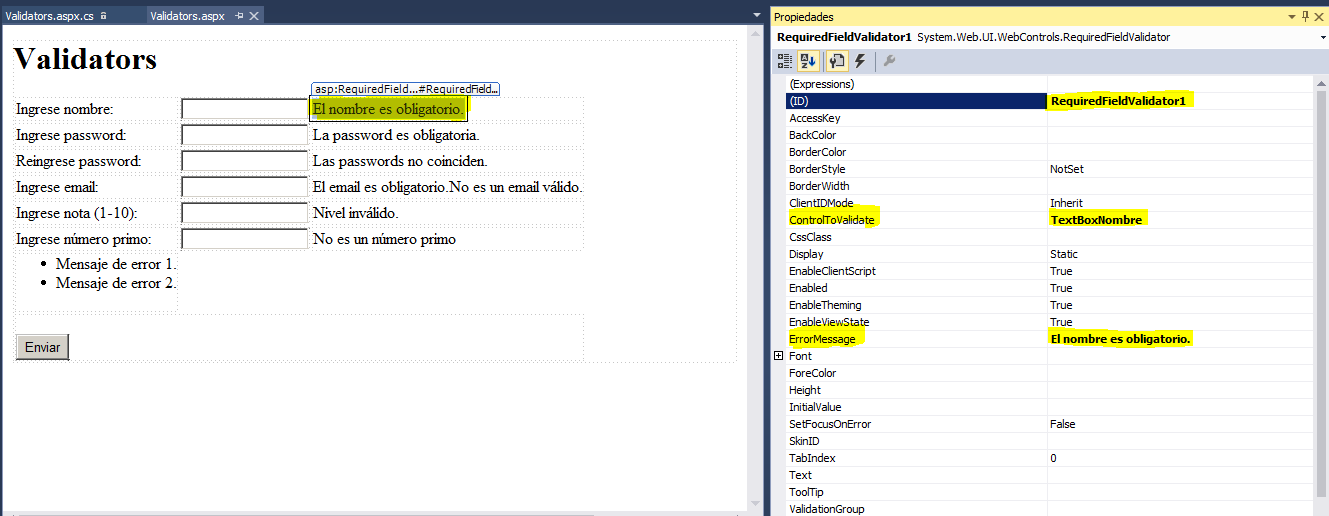


### 

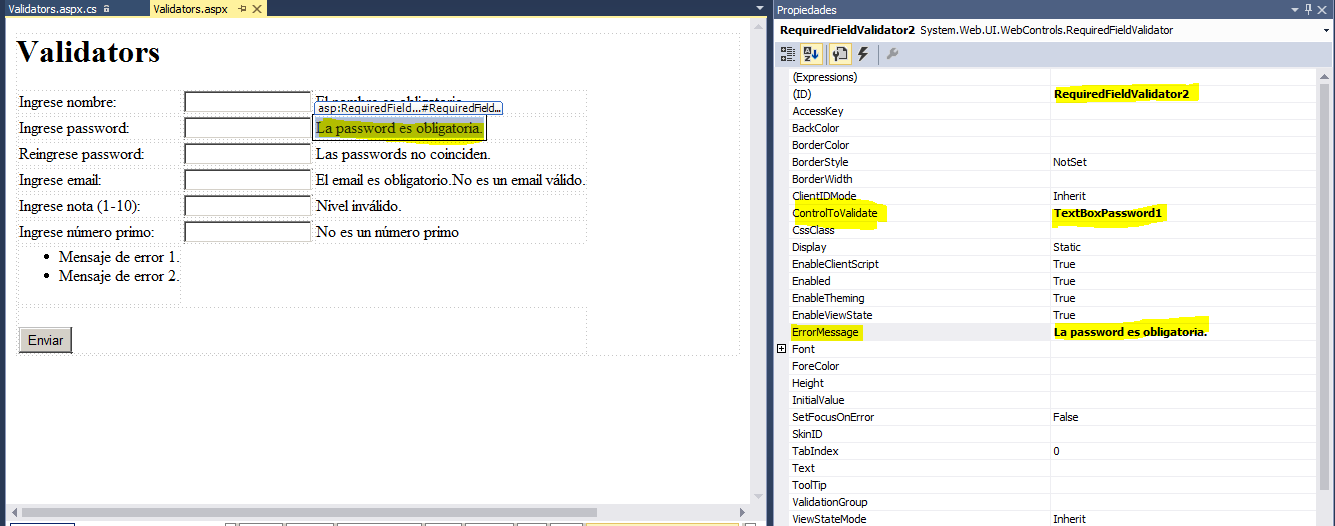
### Tareas

Para ejercitar realice las siguientes actividades:

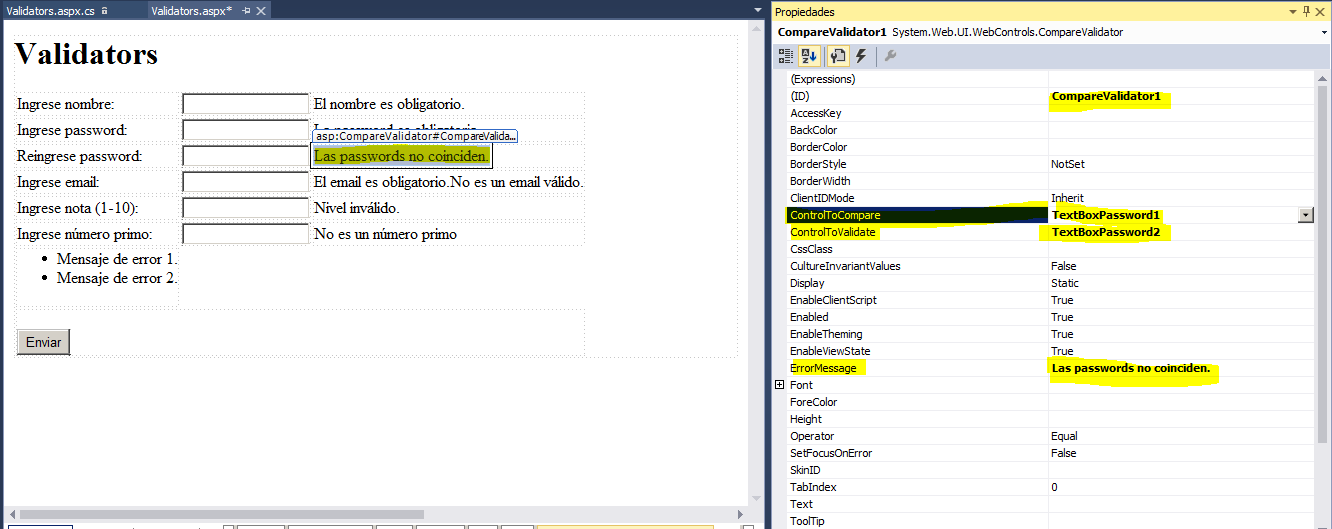
1. En modo diseño del WebForm, ubique el control **RequiredFieldValidator1**, y observe que valida en campo **TextBoxNombre** como carga de dato obligatorio. Y en la propiedad ErrorMessage hay un mensaje que aparecerá cuando el usuario no cargue ningún valor en el textbox.



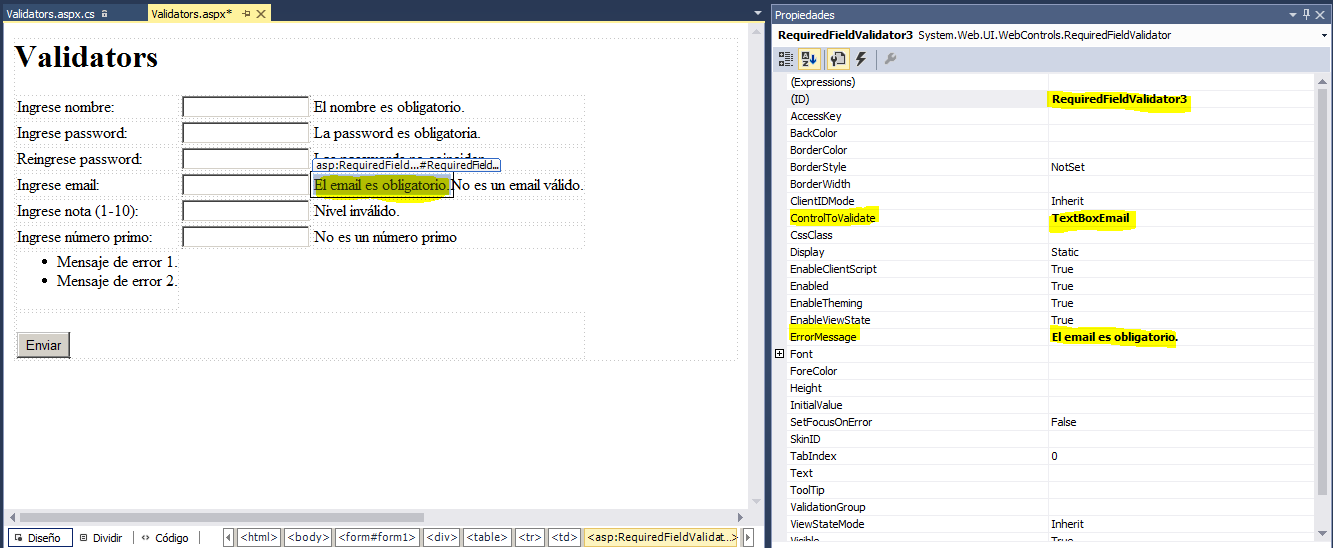
1. Ejecute el proyecto, pruebe evitar cargar un valor en el textbox del nombre. Luego interrumpa la ejecución, deshabilite el validator (propiedad **Enabled** en False), y vuelva a probar dejar el textbox vacío.
2. Pruebe hacer lo mismo que en los puntos anteriores pero con el control **RequiredFieldValidator2** que valida el control **TextBoxPassword1**.

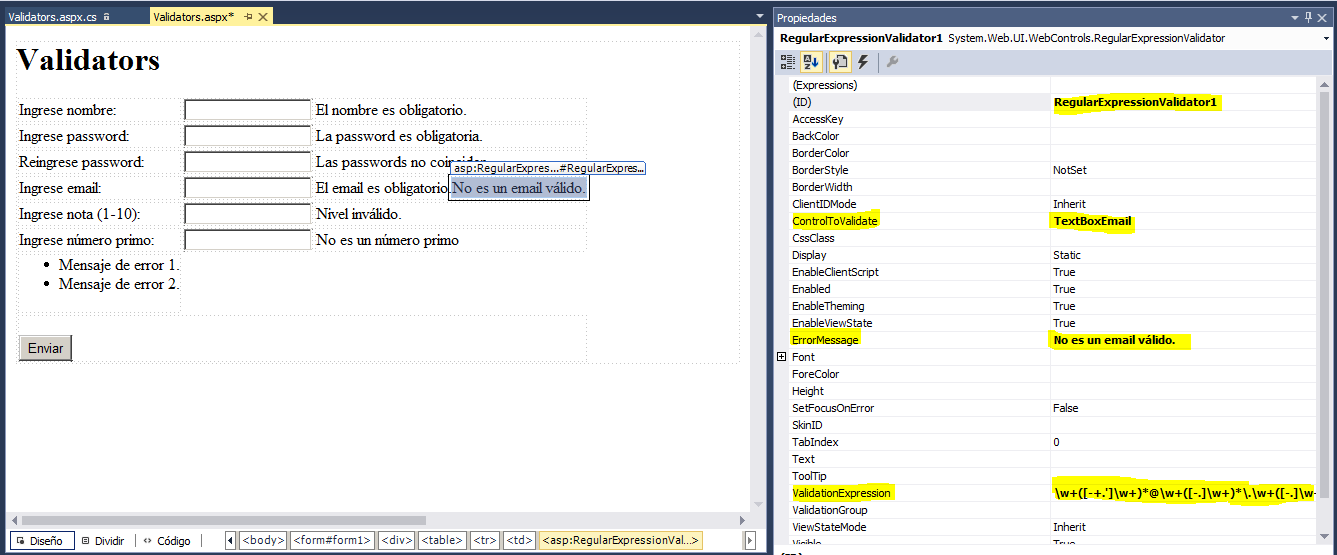


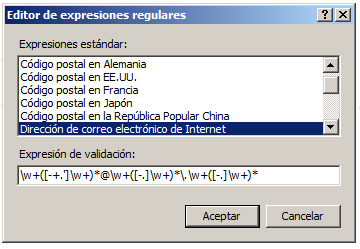
1. En modo diseño del WebForm, ubique el control **CompareValidator1**, que permite comparar el valor de dos controles. Observe que valida el campo **TextBoxPassword2** y que a su vez con compara con el contenido del control **TextBoxPassword1**. También tenemos un mensaje de error en la propiedad **ErrorMessage**. Pruebe ejecutar el proyecto, cargue un valor en el textbox de la password, y luego el mismo valor en el textbox del reingreso de la password. Vuelva a probar la comparación ingresando en ambos textbox dos valores distintos.



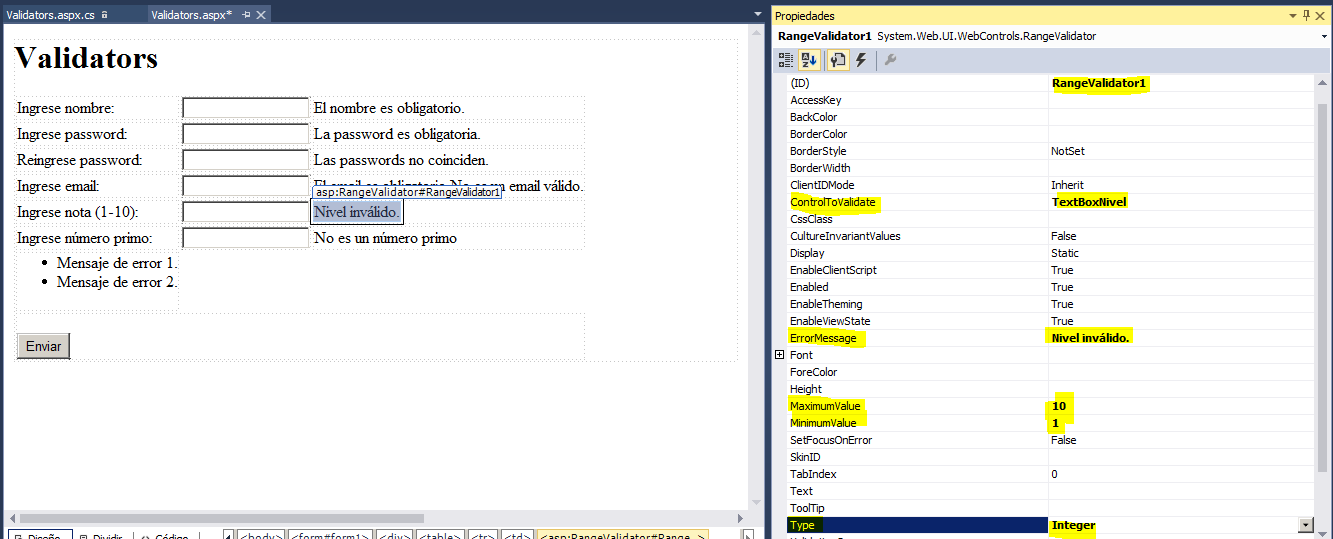
1. En modo diseño del WebForm, ubique los controles **CompareValidator3** y **RegularExpressionValidator1**, los mismos trabajan en conjunto con el cuadro de texto **TextBoxEmail**, que valida la carga de un email y a su vez se asegura que tenga un formato válida vía una expresión regular. También tenemos mensajes de error en ambos controles en la propiedad **ErrorMessage**. Pruebe ejecutar el proyecto, cargue un mail válida en el textbox. Vuelva a probar la carga escribiendo un texto que no represente un email.



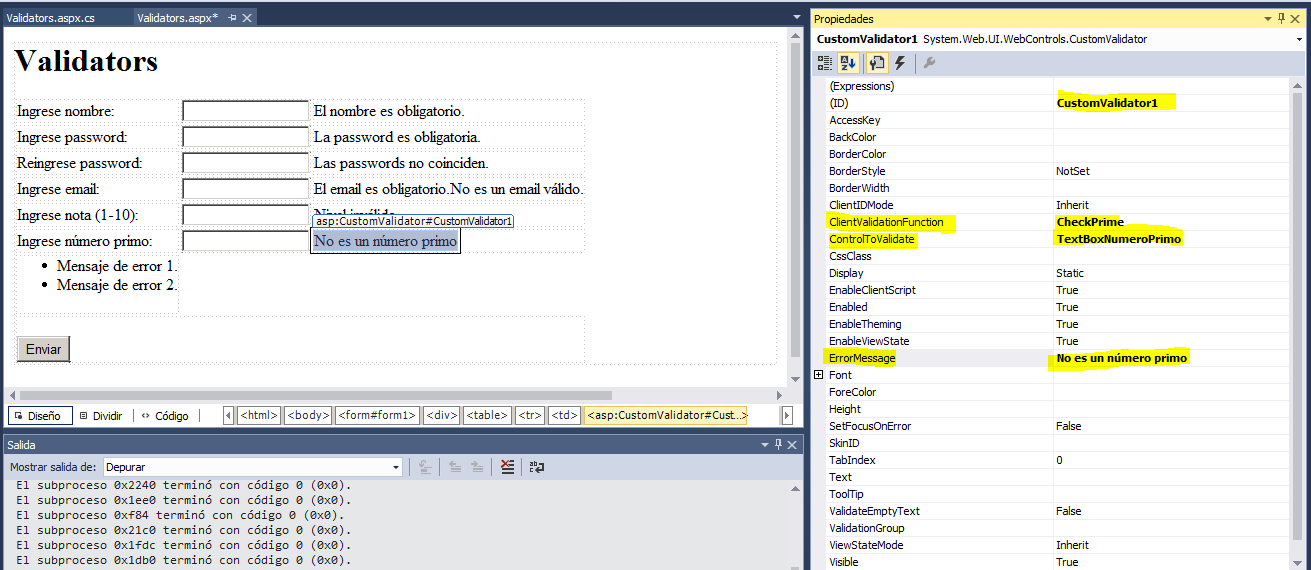




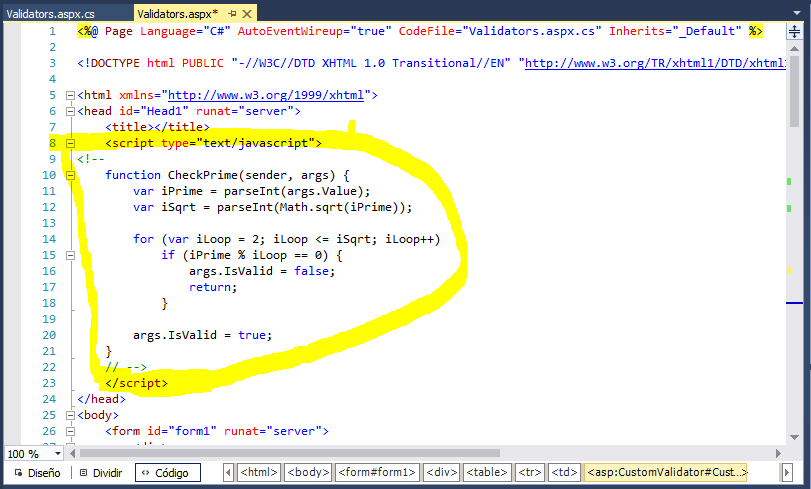
1. Respecto de las expresiones regulares, existen varias precargadas para las máscaras más usuales como direcciones de email, números telefónicos internacionales, formatos de códigos postales, etc. También aquí puede crear expresiones regulares personalizadas (para más detalles consulte el siguiente link https://docs.microsoft.com/es-es/dotnet/standard/base-types/regular-expression-language-quick-reference).
2. En modo diseño del WebForm, ubique los controles **RangeValidator1** se usa para controlar rangos de valores. Observe que valida el textbox **TextBoxNivel**, y también muestra un mensaje de error. Observe además de los límites del rango, que se puede cambiar el tipo de dato del rango. Pruebe ejecutar el proyecto cargando valores dentro y fuera del rango. Lo puede usar también para validar rangos de fechas, de importes y de caracteres.



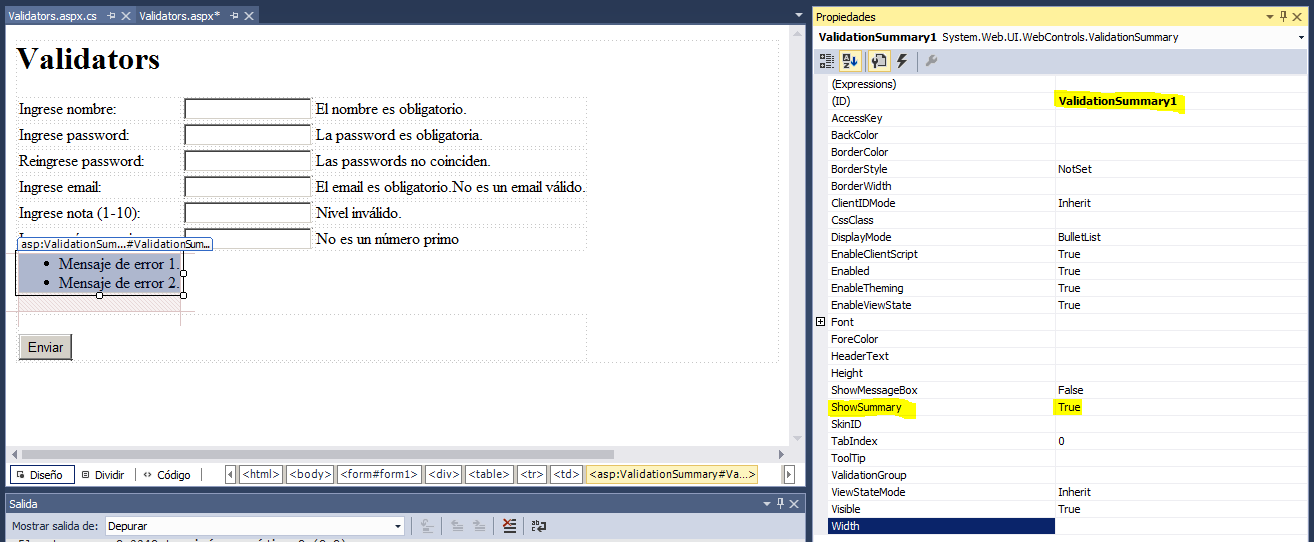
1. En modo diseño del WebForm, ubique los controles **CustomValidator1**se usa para realizar una validación personalizada en el cliente. Observe que valida el textbox **TextBoxNumeroPrimo**, e indica que se valide con la función **CheckPrime.** También muestra un mensaje de error. Pruebe ejecutar el proyecto cargando un número primo y uno que no sea primo.

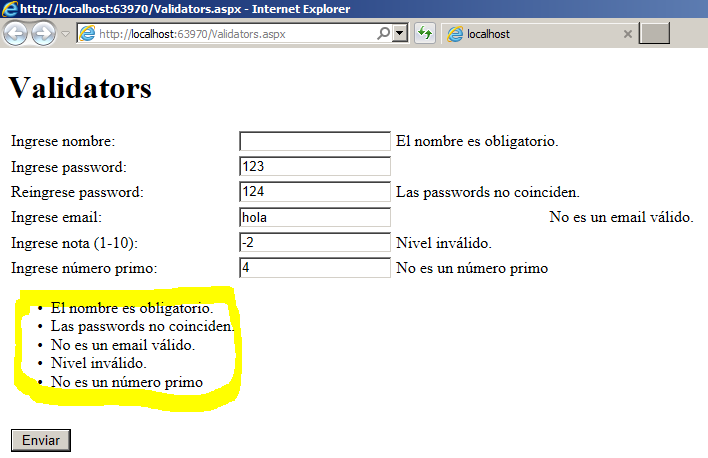


1. La función CheckPrime es una función JavaScript ubicada en la misma página, observe el bloque de scripting:



1. Por último tenemos la posibilidad de mostrar todos los mensajes de error, en una única área de la página y compartida para todos los validadores. Se usa el control **ValidationSummary**. Pruebe ejecutar la aplicación, y luego de cargar todos los valores click en el botón **Enviar**. Vuelva a ejecutar el proyecto pero previo ponga en **False** la propiedad **ShowSummary**.





## Ejercicio 2: Acceso a datos simple

### Introducción

Crearemos un pequeño ejemplo con acceso a datos de ADO.Net y una aplicación de consola. Siga los siguientes pasos para ejercitar este tema.

### Objetivos

Ejercitar con los objetos básicos de ADO.Net.

### Tareas

1. Cree un proyecto de Consola con el nombre **EjemploAdoNet**.
2. Deberá tener disponible la base de datos **Northwind** en Sql Server. En la sección de descargas de la clase 2 de este curso, puede bajar el archivo **Northwind.zip**, que tiene un backup de la base de datos para poder hacerle un restore, o también el script sql para crear la estructura de tablas y llenarla con registros.
3. En el proyecto, en el archivo **Program.cs**, en la sección **Main**, copie el siguiente código. El ejemplo recupera datos de la tabla **Products** de la base de datos, a través de un **comando** ADO.Net que tiene una instrucción **SELECT** con un parámetro. Los resultados se cargan en un objeto **DataReader**:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.SqlClient;

namespace EjemploAdoNet

{

class Program

{

static void Main(string[] args)

{

string connectionString =

"Data Source=(local);Initial Catalog=Northwind;"

+ "Integrated Security=true";

// String sql con la instrucción SELECT y un parámetro para filtrar por precio unitario

string queryString =

"SELECT ProductID, UnitPrice, ProductName from dbo.products "

+ "WHERE UnitPrice > @Precio "

+ "ORDER BY UnitPrice DESC;";

// Indicamos el valor del parámetro

int ValorPrecio = 5;

// Crear y abrir la conexión

using (SqlConnection connection =

new SqlConnection(connectionString))

{

// Crear los objetos Command y Parameter

// En el Command se indica la instrucción SQL de la variable queryString

// Se agrega al Command un nuevo parámetro llamado @Precio

// y se le pasa a ese parámetro, el valor de la variable ValorPrecio

SqlCommand command = new SqlCommand(queryString, connection);

command.Parameters.AddWithValue("@Precio", ValorPrecio);

// Se abre la conexión en un bloque try/catch

// Se crea un objeto DataReader y se ejecuta la instrucción SQL del Command

// El resultado del SELECT quedará en el objeto reader

// Y en un ciclo while, se itera el reader mostrando los registros/campos

try

{

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Console.WriteLine("\t{0}\t{1}\t{2}",

reader[0], reader[1], reader[2]);

}

reader.Close();

Console.ReadKey();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

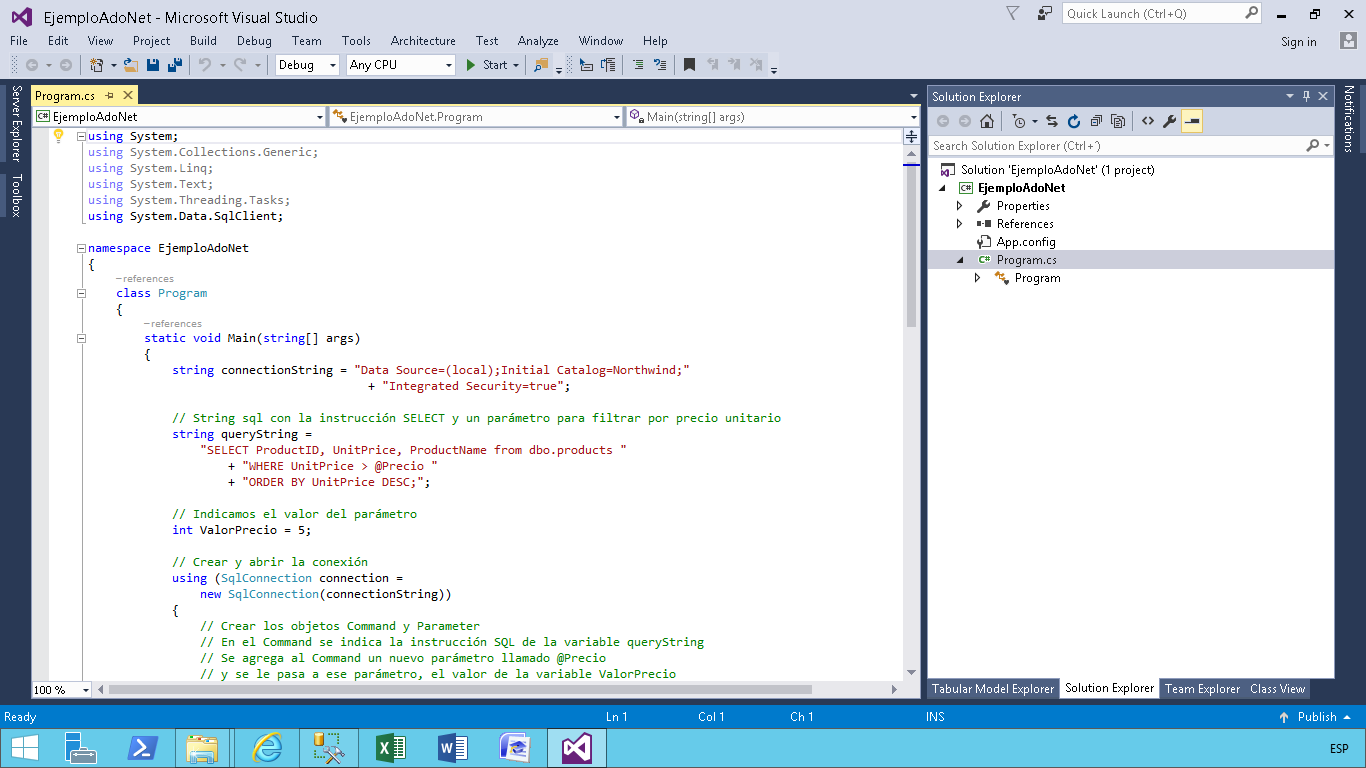
Console.ReadLine();

}

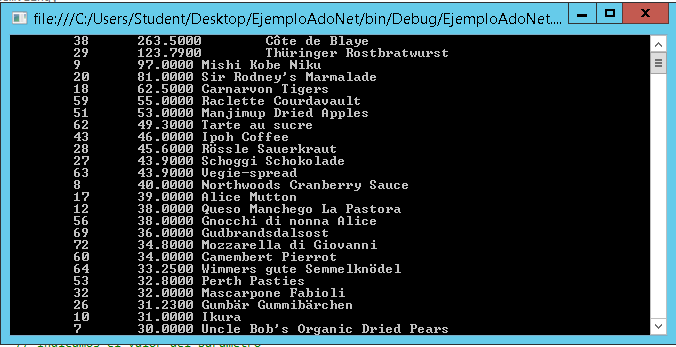
}

}

}



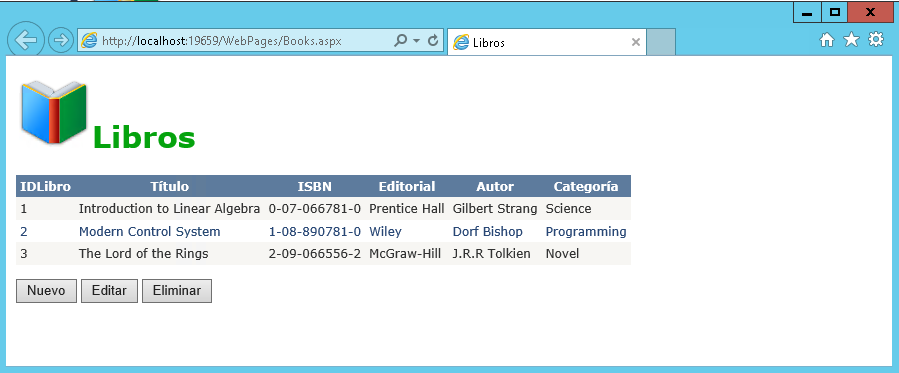
1. Ejecute el proyecto. Verá el resultado del SELECT del DataReader.

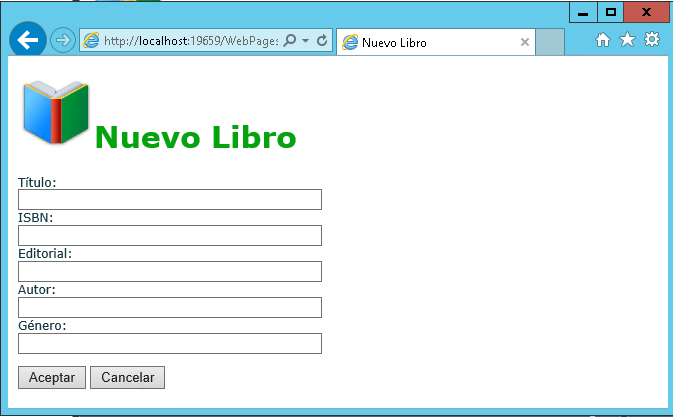


## Ejercicio 3: CRUD con ADO.Net y WebForms

Presentamos para esta práctica el proyecto **CRUDLibros**. Lo encontrará en la sección de descargas de esta clase.

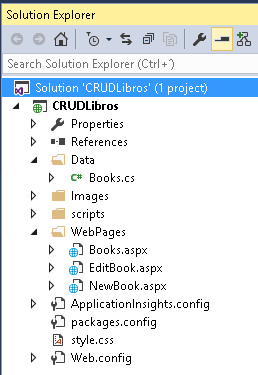
Se trata de una aplicación web que permite consultar, dar de alta, dar de baja y modificar libros (C=create, R=read, U=Update, D=delete).



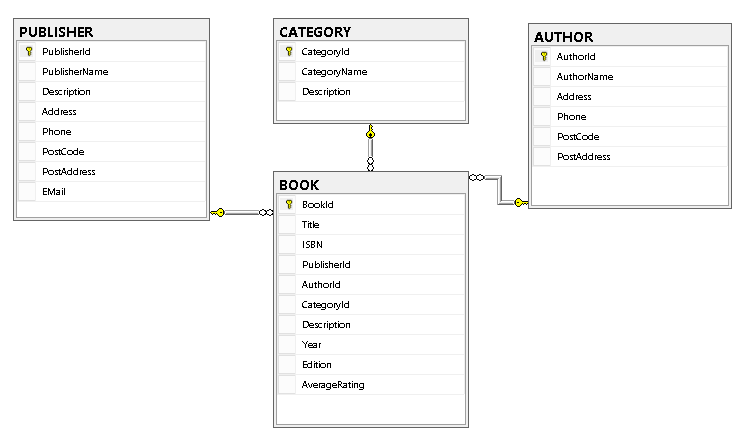


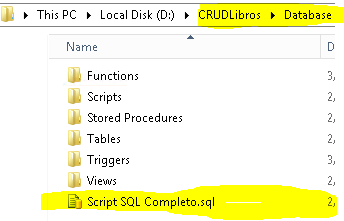


1. Baje el proyecto, descomprima los archivos en una carpeta, y cárguelo en Visual Studio.



1. El proyecto consta de tres WebForms: **Books.aspx**, **EditBook.aspx**, **NewBook.aspx**.
2. La aplicación usa una base de datos Sql Server llamada **BOOKS**. Deberá crearla con el script llamado **Script SQL Completo.sql**.





1. Deberá cambiar el string de conexión para apuntar a la base de datos en la máquina en la que ejercita, archivo **Web.config**:

<connectionStrings>

<add name="LibraryDBConnectionString" connectionString="Data Source=MIA-SQL\;Initial Catalog=BOOKS;Integrated Security=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

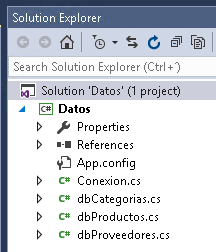
1. Ubique en el proyecto la clase Book (archivo Books.cs). Aquí está definida la clase Libro, con sus atributos, y los métodos para las operaciones del CRUD. Recorra el código de esta clase, el mismo está documentado para que lo pueda comprender mejor, usa los objetos de ADO.NET para llamar a los stored Procedures de SQL, que realizan SELECT, INSERT, UPDATE y DELETE, en las tablas de la base de datos.
2. También analice el código C# de los WebForms: **Books.aspx.cs**, **EditBook.aspx.cs**, **NewBook.aspx.cs**. Este código del lado servidor, llama a los métodos de la clase Book, que interactúan vía ADO.NET con la base de datos.

## Ejercicio 4: Capa de Acceso a datos del Ejercicio Integrador

En el proyecto integrador en capas y en clases anteriores, ya trabajó con la capa de negocio que incluye las entidades de negocio y la lógica de negocio.

Ahora exploraremos la capa de acceso a datos del proyecto. No la probaremos, sólo analizaremos el código dada sus dependencias en el modelo de capas. En las próximas clases verá el proyecto en capas completo.

1. Para explorar el proyecto, de la sección de descargas descargue el archivo Datos.zip. Descomprímalo en una carpeta.
2. Luego cargue el proyecto, deberá dar doble click al archivo Datos.csproj.



1. La capa de acceso a datos consta de cuatro archivos de clase:

|  |  |
| --- | --- |
| Conexion.cs | Tiene tres propiedades protegidas para establecer la conexión a un origen de datos, que son accesibles desde la misma clase **Conexión** y sus clases derivadas.  La propiedad **objConexion** toma la información del string de conexión de una variable de entorno que luego estará en el archivo Web.config de la aplicación completa.  La propiedad **objComando**, deja seteada la conexión y el tipo de comando que se ejecutará. Luego en los métodos de las demás clases se completará. |
| dbCategorias.cs | Hereda de la clase Conexión. Tiene un método para listar todas las categorías existentes. Usa las propiedades **objConexion** y **objComando**, recupera las de la base de datos a través de un data reader y retorna una lista de entidades tipo **Categoria**. |
| dbProductos.cs | Hereda de la clase Conexión. Tiene un método para listar todos los productos existentes con diversos filtros. Usa las propiedades **objConexion** y **objComando**, recupera las de la base de datos a través de un data reader y retorna una lista de entidades tipo **Producto** y **VerProducto**. También tiene métodos para agregar, actualizar y eliminar un producto. |
| dbProveedores.cs | Hereda de la clase Conexión. Tiene un método para listar todos los proveedores existentes. Usa las propiedades **objConexion** y **objComando**, recupera las de la base de datos a través de un data reader y retorna una lista de entidades tipo **Proveedor**. |