

Validación personalizada: métodos, servicios y clases ValidationRule

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Reglas de negocio que las reglas integradas no pueden expresar: métodos de validación que devuelven un string, un servicio ContactValidator que se puede probar sin la interfaz, una clase ValidationRule personalizada con OnValidating, OnValidated y OnControlCreated, y reglas entre campos que marcan todos los controles implicados. Qué significa esto para un desarrollador que conoce C# básico y empieza a desarrollar formularios con Wisej.NET, y cómo leer este módulo.



Enfoque de producción

Reglas de negocio que las reglas integradas no pueden expresar: métodos de validación que devuelven un string, un servicio ContactValidator que se puede probar sin la interfaz, una clase ValidationRule personalizada con OnValidating, OnValidated y OnControlCreated, y reglas entre campos que marcan todos los controles implicados.

1. El problema en producción

Después del Módulo 3, el formulario de alta de ValidationClinic sabe que un correo electrónico necesita una @ y que una edad debe ser un número entero. Pero sigue guardando sin problema un contacto cuya fecha de inicio es posterior a la fecha de fin, o una fecha de nacimiento que hace que la persona tenga once años. Las reglas integradas comprueban la *forma* de un valor; las aplicaciones reales también necesitan reglas sobre el *significado*, y algunas de esas reglas implican a más de un control. Este módulo da a esas reglas un lugar adecuado, para que se escriban una sola vez, se muestren con claridad y nunca se omitan al guardar.

2. Conceptos clave

Concepto	Por qué importa
método de validación personalizado	Un método del formulario que responde a una única pregunta y devuelve una cadena vacía cuando los datos son válidos, o un mensaje claro cuando no lo son.
servicio de validación	Una clase como <code>ContactValidator</code> que comprueba un modelo sin tocar ningún control, de modo que las mismas reglas se ejecutan desde el formulario, una cuadrícula o una prueba unitaria.
record <code>ValidationMessage</code>	Un tipo de resultado mínimo que contiene un nombre de <code>Field</code> y un <code>Message</code> ; el servicio devuelve una lista de ellos en lugar de establecer los errores por sí mismo.
<code>ApplyModelValidation</code>	El método del formulario que borra los errores antiguos, asocia cada nombre de campo a su control con <code>errorProvider.SetError</code> , rellena el resumen e informa de si todo ha pasado.
<code>ValidationRule</code>	La clase base de Wisej.NET que hay detrás de cada regla del extensor Validation; derivar de ella convierte tu propia comprobación en una regla que puedes asociar a cualquier control.
<code>OnValidating</code> / <code>OnValidated</code> / <code>OnControlCreated</code>	Las tres sobrescrituras de una clase de regla: comprobar el valor y devolver true o false, actuar después de que un valor haya pasado y preparar lo necesario una vez que el control existe en el navegador.
regla entre campos	Una regla cuya respuesta depende de dos o más controles, como que la fecha de inicio sea anterior a la de fin, así que su error corresponde a todos los controles implicados.

3. Desarrollo de la lección

Empieza con un método que devuelve un mensaje

La regla personalizada más sencilla es un método normal de C#. Lee los valores que necesita, hace una pregunta y devuelve un string: vacío cuando todo está bien, y una frase con la que el usuario puede actuar cuando no lo está. `ValidateStartEndDates`, en el laboratorio, lee `dtpStart.Value` y `dtpEnd.Value`, devuelve «Both dates are required.» cuando falta alguna, «Start date must be before end date.» cuando el orden es incorrecto, y "" en los demás casos.

Fíjate en lo que el método *no* hace: no toca el `ErrorProvider`. Un segundo método pequeño recibe el texto devuelto y lo pasa a `errorProvider.SetError` para cada selector. Como un mensaje vacío borra un error, las mismas dos líneas muestran el problema y lo eliminan cuando se corrige. Mantener la decisión (el string) separada de la visualización (las llamadas a `SetError`) es el hábito que facilita el siguiente paso.

Traslada las reglas repetidas a un servicio de validación

En cuanto la misma comprobación se necesita en un segundo lugar, por ejemplo el formulario de alta y un cuadro de diálogo de alta rápida, un método en un solo formulario ya no basta. El laboratorio traslada las reglas del nombre, el correo electrónico y la fecha de nacimiento a una clase `ContactValidator`. Su método `Validate` recibe un `ContactEditModel`, un objeto sencillo con las propiedades `Name`, `Email` y `BirthDate`, y devuelve una lista de records `ValidationMessage(Field, Message)`: «Name is required.», «Email must contain @.», «Birth date cannot be in the future.».

El servicio no sabe nada de cuadros de texto, así que puedes llamarlo desde una prueba con un modelo creado a mano y comprobar la lista que devuelve, sin navegador y sin ninguna sesión de Wisej.NET en marcha. El formulario solo necesita entonces un adaptador, `ApplyModelValidation`. Este llama a `errorProvider.Clear()`, ejecuta el validador, usa un `switch` para convertir `"Name"`, `"Email"` y `"BirthDate"` en `txtName`, `txtEmail` y `dtpBirthDate`, establece cada error, une todos los mensajes en `validationSummaryLabel` y devuelve true solo cuando la lista está vacía. Un mensaje sin control asociado sigue llegando al resumen, así que no se pierde nada. El Módulo 5 construirá el modelo mediante el enlace a datos; aquí lo rellenas tú a partir de los controles.

Crea una clase derivada de ValidationRule

Algunas reglas pertenecen a un único tipo de control y deberían poder llevarse de un formulario a otro como las integradas. Para esas, deriva de `ValidationRule`. `MinimumAgeValidationRule` tiene una propiedad `MinimumAge` cuyo valor predeterminado es 18, y su constructor llama a `base("MinimumAge", "The person is too young.")`, lo que da a la regla su nombre y el mensaje de valor no válido que muestra el extensor cuando la regla falla.

La asocias exactamente igual que las reglas del Módulo 3, pasando una instancia a `validation.SetValidationRules` para `dtpBirthDate`. A partir de ese momento, el extensor Validation la ejecuta cada vez que se valida el selector, cuando el usuario sale del campo y cuando `ValidateChildren()` se ejecuta al guardar, y envía el mensaje al mismo `ErrorProvider` que las demás reglas.

Para qué sirven OnValidating, OnValidated y OnControlCreated

`OnValidating(Control control)` es el corazón de la regla. Devuelve `true` cuando el valor es válido y `false` cuando no lo es; no devuelve un mensaje, porque el mensaje se proporcionó al constructor. La versión del laboratorio comprueba que el control es un `DateTimePicker` cuyo `Value` es un `DateTime`, calcula la edad y resta un año si el cumpleaños todavía no ha llegado este año:

```
public override bool OnValidating(Control control)
{
    if (control is DateTimePicker picker && picker.Value is DateTime birthDate)
    {
        var today = DateTime.Today;
        var age = today.Year - birthDate.Year;
        if (birthDate.Date > today.AddYears(-age))
            age--;
        return age ≥ MinimumAge;
    }
    return true;    // nothing to check: Required decides about empty values
}
```

Las otras dos sobrescrituras son opcionales. `OnValidated` se ejecuta después de que un valor haya pasado, lo que la convierte en el lugar para el trabajo posterior, como dar formato al valor aceptado. `OnControlCreated` se ejecuta una vez creado el control en el cliente, así que cualquier configuración que necesite que exista el widget del navegador va ahí y no en el constructor. Para una regla de nivel principiante, lo normal es sobrescribir solo `OnValidating`.

Reglas entre campos que no confunden a los usuarios

Una fecha de inicio posterior a la de fin no es culpa de uno solo de los selectores, así que no culpes a uno de ellos. Establece el mismo mensaje en `dtpStart` y en `dtpEnd`, añade una línea al resumen y borra ambos en cuanto el orden sea correcto. Evita `e.Cancel = true` en este tipo de regla: bloquearía el foco en el primer selector cuando la corrección puede estar en el segundo.

Después convierte Save en la última barrera. `btnSave_Click` ejecuta primero las reglas de campo con `ValidateChildren()`, lo que incluye `MinimumAgeValidationRule`. Si eso pasa, construye el modelo y llama a `ApplyModelValidation`, y a continuación ejecuta la comprobación de fechas, porque de lo contrario `errorProvider.Clear()` borraría los errores de fechas. Solo cuando todas las capas pasan llama al repositorio. Una regla que falla es un aviso normal del producto, no una excepción, así que la base de datos nunca se abre para datos que ya sabes que son incorrectos.



Regla de las capas de validación

Nombra la capa antes de escribir la comprobación. La validación de campo protege la edición y se ejecuta cuando el usuario sale de un control; la validación del modelo protege los datos y ejecuta las mismas reglas desde el formulario, la cuadrícula y cualquier servicio; la validación al guardar protege la base de datos y vuelve a ejecutar todas las capas antes de escribir nada. Muestra cada error cerca de su campo y en un resumen visible, con un lenguaje claro que diga qué hacer a continuación, y mantén los errores que el usuario puede corregir separados de los fallos del sistema.



Objetivos de aprendizaje

Al final de este módulo serás capaz de:

- ✓ Escribir métodos de validación personalizados para reglas de negocio de nivel principiante.
- ✓ Trasladar las reglas repetidas a un servicio de validación reutilizable.

- ✓ Crear una clase personalizada derivada de `ValidationRule`.
- ✓ Comprender `OnValidating`, `OnValidated` y `OnControlCreated`.
- ✓ Aplicar validación entre campos sin confundir a los usuarios.

Guía del laboratorio / examen

Qué vas a construir en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Añade validación personalizada a ValidationClinic en tres pasos. Escribe ValidateStartEndDates como un método que devuelve una cadena vacía cuando todo es válido y un mensaje cuando la fecha de inicio es posterior a la fecha de fin, y establece ese mensaje tanto en dtpStart como en dtpEnd. Traslada las reglas del nombre, el correo electrónico y la fecha de nacimiento a un servicio ContactValidator que devuelve una lista de records ValidationMessage, y asocia cada nombre de campo a su control en ApplyModelValidation. Después crea MinimumAgeValidationRule derivando de ValidationRule con una propiedad MinimumAge, sobrescribe OnValidating para calcular la edad a partir del DateTimePicker, asíocia a dtpBirthDate mediante el extensor Validation y ejecuta el servicio en btnSave_Click antes de guardar nada.

Punto de partida

Abre ValidationClinic tal como lo dejaste después del Módulo 3: el formulario de alta tiene un extensor Validation y un `ErrorProvider` que comprueban el correo electrónico, la edad, el teléfono, el límite de crédito y el código de cliente, además de un `validationSummaryLabel`. Los selectores `dtpBirthDate`, `dtpStart` y `dtpEnd` todavía no tienen reglas, y `btnSave_Click` guarda todo lo que deja pasar `ValidateChildren()`.

Enfoque sugerido

- ✓ Escribe `ValidateStartEndDates` para que devuelva "" o un mensaje, y después un pequeño método auxiliar que pase ese mismo string a `errorProvider.SetError` tanto para `dtpStart` como para `dtpEnd`. Prueba un orden incorrecto, corrígelo y comprueba que los dos iconos aparecen y los dos desaparecen.
- ✓ Crea un record `ValidationMessage(Field, Message)` y una clase `ContactValidator` cuyo método `Validate` compruebe en un `ContactEditModel` si hay un nombre vacío, un correo electrónico ausente o sin @ y una fecha de nacimiento futura. No debe hacer referencia a ningún control.
- ✓ Añade `ApplyModelValidation` al formulario: borra el `ErrorProvider`, ejecuta el validador, asocia `"Name"`, `"Email"` y `"BirthDate"` con sus controles, rellena la etiqueta de resumen y devuelve si la lista estaba vacía.
- ✓ Crea `MinimumAgeValidationRule` derivando de `ValidationRule`, llama a `base("MinimumAge", "The person is too young.")`, sobrescribe `OnValidating` para calcular la edad y asocia una instancia a `dtpBirthDate` con `validation.SetValidationRules`.
- ✓ Reescribe `btnSave_Click` por capas: detente si `ValidateChildren()` falla, detente si `ApplyModelValidation` o la comprobación de fechas fallan y, solo entonces, llama al repositorio dentro de un `try/catch` para los fallos reales del sistema. Añade a tu archivo del laboratorio una breve nota que nombre cada API que has usado.

Entregables requeridos

- ✓ Método `ValidateStartEndDates` que establece el mismo error en los dos selectores de fecha y borra ambos cuando se corrige
- ✓ Servicio `ContactValidator` que devuelve records `ValidationMessage` para el nombre, el correo electrónico y la fecha de nacimiento
- ✓ Método `ApplyModelValidation` que asocia cada nombre de campo a su control y al resumen
- ✓ Clase `MinimumAgeValidationRule` que sobrescribe `OnValidating`, asociada a `BirthDate`
- ✓ Controlador de `Save` que ejecuta las reglas de campo, después el servicio validador y solo entonces guarda

Preguntas de repaso

- ✓ La comprobación de la edad mínima podría estar en `ContactValidator` o en `MinimumAgeValidationRule`. ¿Qué percibe de forma distinta el usuario en cada caso y cuándo querrías tener ambas?
- ✓ Tu controlador de `Save` llama a `ApplyModelValidation` después de establecer los errores de fechas, y los iconos de las fechas nunca aparecen. Explica por qué y qué orden lo soluciona.
- ✓ ¿Cómo demostrarías que `ContactValidator` rechaza un correo electrónico sin @ sin iniciar la aplicación ni abrir un navegador?



Regla de las capas de validación

Nombra la capa antes de escribir la comprobación. La validación de campo protege la edición y se ejecuta cuando el usuario sale de un control; la validación del modelo protege los datos y ejecuta las mismas reglas desde el formulario, la cuadrícula y cualquier servicio; la validación al guardar protege la base de datos y vuelve a ejecutar todas las capas antes de escribir nada. Muestra cada error cerca de su campo y en un resumen visible, con un lenguaje claro que diga qué hacer a continuación, y mantén los errores que el usuario puede corregir separados de los fallos del sistema.