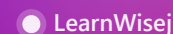


Fundamentos de la validación en Wisej.NET

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Qué significa validar en una interfaz de Wisej.NET que se ejecuta en el servidor: el ciclo de vida de Validating y Validated, cancelar la entrada no válida con e.Cancel, el comportamiento del foco con AutoValidate, CausesValidation en los botones Cancel y ValidateChildren como protección de Save. 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

Qué significa validar en una interfaz de Wisej.NET que se ejecuta en el servidor: el ciclo de vida de Validating y Validated, cancelar la entrada no válida con e.Cancel, el comportamiento del foco con AutoValidate, CausesValidation en los botones Cancel y ValidateChildren como protección de Save.

1. El problema en producción

ValidationClinic es la aplicación de ejemplo que construyes tú mismo a lo largo de este curso: la Guía del laboratorio / examen de este módulo explica exactamente cómo empezar, y el Ejercicio de programación con IA puede hacer que un asistente la monte por ti. Su formulario de alta de clientes acepta ahora cualquier cosa: un nombre vacío, un correo electrónico sin @ o un clic en Save en un formulario que nadie ha rellenado. La primera idea que tienen casi todos los principiantes es poner una instrucción `if` dentro del botón Save, pero eso no da ningún aviso al usuario mientras trabaja y convierte Save en un muro de errores al final. Este módulo coloca la validación donde le corresponde en un formulario de Wisej.NET: en cada editor cuando el usuario sale de él, y una vez más cuando el usuario hace clic en Save.

2. Conceptos clave

Concepto	Por qué importa
validación	La decisión de si un valor es aceptable antes de que la aplicación lo use, seguida de un aviso visible y, cuando el valor es incorrecto, de la negativa a continuar.
<code>Validating</code>	El evento que desencadena un control cuando el usuario intenta salir de él (o cuando el código pide una validación); su controlador lee el valor y es el único lugar donde se decide aceptar o rechazar.
<code>CancelEventArgs.Cancel</code>	El indicador que un controlador <code>Validating</code> establece en <code>true</code> para rechazar el valor: el control cuenta como no válido, el foco puede retenerse en él y <code>ValidateChildren()</code> informa del fallo.
<code>Validated</code>	El evento que se desencadena solo después de que <code>Validating</code> haya terminado sin cancelar; el lugar para tareas de limpieza como recortar texto, nunca para guardar.
<code>AutoValidate</code>	Una propiedad de un formulario, una página o un control de usuario que decide qué ocurre con el foco cuando un control hijo no supera la validación.
<code>CausesValidation</code>	Un interruptor por control; establecido en <code>false</code> en un botón como Cancel, hacer clic en él no valida el campo que el usuario está abandonando.
<code>ValidateChildren()</code>	Un método del contenedor que ejecuta la validación de sus controles hijos, incluidos los campos que el usuario nunca visitó, y devuelve <code>false</code> si algún controlador canceló.

3. Desarrollo de la lección

La validación en una interfaz que se ejecuta en el servidor

En Wisej.NET construyes un formulario como construirías cualquier clase de C#: un `CustomerIntakeForm` con campos llamados `txtName`, `txtEmail` y `btnSave`. Ese objeto formulario y sus controles viven en el servidor, dentro de la sesión del usuario. El navegador los dibuja e informa de lo que hace el usuario (escribir, hacer clic, mover el foco), y tus controladores de eventos se ejecutan como C# normal en el servidor. Así, cuando el usuario pulsa Tab para salir de `txtEmail`, el navegador comunica al servidor que el foco se mueve, el servidor desencadena el evento de validación de `txtEmail`, tu controlador lee `txtEmail.Text` y el resultado, como un icono de error o el foco retenido en su sitio, se devuelve y se dibuja.

Esto tiene dos consecuencias prácticas. La decisión la toma tu código de servidor, así que el navegador no puede saltársela. Y cada cambio de foco con validación es un viaje de ida y vuelta, así que un controlador debe ser rápido y predecible: mirar este único campo y después establecer o borrar un mensaje. Una consulta a un servicio remoto con cada pulsación de Tab hace que el formulario parezca lento; reserva ese tipo de comprobación para Save. Por último, un mensaje de validación es un texto del producto escrito para el usuario, como «Enter the customer's name.», no un mensaje de excepción.

Validating decide, Validated pone orden

El ciclo de vida tiene un orden fijo. El usuario edita un control y después intenta salir de él, hacer clic en un botón que valida o guardar el formulario. El control desencadena `Validating`. Tu controlador examina el valor: si no es válido, establece un mensaje y cancela; si es válido, borra cualquier mensaje antiguo. Solo cuando `Validating` no ha cancelado, el control desencadena `Validated`.

Mantén separados los papeles. `Validating` es donde se toma la decisión. `Validated` es para la limpieza de un valor que ya has aceptado, como recortar los espacios que un usuario ha escrito alrededor de un nombre. Nunca guardes desde `Validated`: un usuario puede validar el mismo campo muchas veces antes de hacer clic en Save. Puedes asociar ambos controladores desde la lista de eventos de la ventana Propiedades de Visual Studio, o en el constructor con `+=`.

```
private void txtName_Validating(object sender, CancelEventArgs e)
{
    if (string.IsNullOrEmpty(txtName.Text))
    {
        e.Cancel = true;
        errorProvider.SetError(txtName, "Enter the customer's name.");
        return;
    }
    errorProvider.SetError(txtName, "");
}

private void txtName_Validated(object sender, EventArgs e)
{
    txtName.Text = txtName.Text.Trim();
}
```

Rechazar un valor con e.Cancel

Un controlador `Validating` recibe un `CancelEventArgs` como segundo parámetro. Establecer `e.Cancel = true` es el rechazo; el mensaje de `ErrorProvider` es la explicación, y necesitas ambos. Un mensaje sin `e.Cancel` es el error de principiante más habitual: el campo parece no válido, pero el valor se acepta, el foco avanza y `ValidateChildren()` informa más tarde de que todo está bien. Cancelar sin mensaje es el problema contrario: el usuario queda bloqueado y no sabe por qué.

Para el mensaje, añade un `ErrorProvider` al formulario y establece su `ContainerControl` en el formulario. `errorProvider.SetError(txtName, "Enter the customer's name.")` muestra un icono de error junto al control con ese texto, y `SetError(txtName, "")` lo elimina. La rama que borra importa: sin ella, un campo corregido sigue mostrando su error antiguo. El Módulo 2 trata `ErrorProvider` en profundidad. Además, no lances excepciones para errores corrientes como una @ que falta. Las excepciones son para fallos excepcionales, como un repositorio al que no se puede acceder.

Elegir un modo de AutoValidate

Qué ocurre con el foco cuando un controlador cancela lo decide la propiedad `AutoValidate` del contenedor: el formulario, la página o el control de usuario que contiene los editores. Hay cuatro opciones:

- `EnablePreventFocusChange`: el foco se queda en el campo no válido hasta que se corrige.

- `EnableAllowFocusChange` : el foco puede moverse y el error sigue visible; Save sigue negándose.
- `Disable` : no se valida nada al cambiar el foco; la validación solo se ejecuta cuando el código llama a `Validate()` o a `ValidateChildren()` .
- `Inherit` : el contenedor usa la configuración de su padre.

El formulario del laboratorio tiene dos campos obligatorios, así que `EnablePreventFocusChange` es una opción clara y adecuada para principiantes. También es estricta: si el nombre está vacío y el usuario hace clic en Cancel, el TextBox no válido puede negarse a ceder el foco y Cancel parece no funcionar. La solución es `btnCancel.CausesValidation = false` . Hacer clic en un control cuyo `CausesValidation` es `false` no valida el campo que se abandona, así que el controlador de Cancel se ejecuta y el usuario siempre puede salir. Save mantiene el valor predeterminado `true` . Tu nota del laboratorio recoge qué modo has elegido y por qué Cancel sigue funcionando.

ValidateChildren() como protección de Save

La validación de campo solo se ejecuta en los campos que el usuario abandona realmente. Un usuario puede escribir un nombre y hacer clic en Save sin haber entrado nunca en `txtEmail` , así que su controlador `Validating` nunca se ha ejecutado. Por eso la primera instrucción de `btnSave_Click` es `if (!this.ValidateChildren()) return; . ValidateChildren()` desencadena la validación de los controles hijos del formulario, visitados o no, muestra todos los mensajes que establecen sus controladores y devuelve `false` si alguno de ellos ha cancelado. Una sobrecarga recibe `ValidationConstraints` para limitar qué hijos se comprueban, por ejemplo solo los controles habilitados, los seleccionables, los que tienen tabulación o los hijos directos, lo que ayuda cuando una parte del formulario está oculta o deshabilitada.

`Validate()` es la herramienta más pequeña: valida el control que está perdiendo el foco, lo que resulta útil cuando `AutoValidate` es `Disable` y quieres comprobar solo el campo actual. Cuando `ValidateChildren()` pase, coloca el guardado real dentro de `try / catch` , para que una escritura fallida muestre un mensaje de sistema comprensible y nunca se confunda con un error de entrada que el usuario puede corregir. La validación de campo protege la edición; la validación al guardar protege el guardado. Necesitas ambas.



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:

- ✓ Explicar qué significa validar en una interfaz de Wisej.NET que se ejecuta en el servidor.

- ✓ Gestionar Validating y Validated en los controles de edición.
- ✓ Cancelar la entrada no válida estableciendo `CancelEventArgs.Cancel`.
- ✓ Elegir un modo de `AutoValidate` para un formulario o un control de usuario.
- ✓ Llamar a `Validate` y a `ValidateChildren` en el momento adecuado.

Guía del laboratorio / examen

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



Laboratorio práctico

Crea el formulario de alta de clientes de ValidationClinic con los TextBox txtName y txtEmail, un botón Save y un botón Cancel. Asocia controladores Validating que establezcan `e.Cancel = true` y un mensaje de ErrorProvider cuando el nombre esté vacío o el correo electrónico no contenga @, que borren el mensaje cuando el valor sea válido, y recorta los espacios del nombre en su controlador Validated. Establece el modo AutoValidate del formulario, asegúrate de que Cancel no provoque validación para que el usuario siempre pueda salir, y llama a `ValidateChildren()` al principio de `btnSave_Click` para que Save se detenga cuando algún campo no sea válido. Termina con una breve nota del laboratorio que explique el comportamiento del foco que has elegido.

Punto de partida

ValidationClinic es la aplicación que construyes tú mismo, módulo a módulo: no se descarga nada ni se instala ninguna solución inicial por ti. Crea en Visual Studio un proyecto Wisej.NET 4 nuevo llamado **ValidationClinic**, dale una página principal, una cuadrícula de contactos editable y un sencillo repositorio de contactos en memoria detrás de esa cuadrícula, y no valides nada todavía; todos los módulos siguientes amplían este mismo proyecto. Dos atajos si prefieres no montar el esqueleto a mano: la sección **Ejercicio de programación con IA** de este módulo entrega a ChatGPT o Claude un prompt ya preparado que construye la solución completa a partir de la especificación del módulo, y nuestra propia versión terminada de cada módulo de este curso está en el repositorio público de ejemplos **LearnWisej-Samples**, con una carpeta por módulo: **Module 1** es ese esqueleto más la solución de este laboratorio. En este laboratorio añades el formulario de alta de clientes y le das sus dos primeras capas de validación: la validación de campo en cada editor y la validación al guardar en el botón Save.

Enfoque sugerido

- ✓ Añade un `CustomerIntakeForm` y usa el diseñador para colocar `txtName`, `txtEmail`, `btnSave`, `btnCancel` y una etiqueta `lblStatus`. Arrastra un `ErrorProvider` al formulario y establece su `ContainerControl` en el formulario.
- ✓ Gestiona `txtName.Validating` y `txtEmail.Validating`. En cada uno, comprueba el valor; cuando sea incorrecto, establece `e.Cancel = true` y llama a `errorProvider.SetError` con un mensaje claro, y cuando sea correcto, llama a `SetError` con una cadena vacía.
- ✓ Gestiona `txtName.Validated` y recorta ahí el texto, para que el valor solo se limpie después de haber sido aceptado.
- ✓ Establece el `AutoValidate` del formulario y después `btnCancel.CausesValidation = false`. Ejecuta el formulario, deja el nombre vacío y haz clic en Cancel para demostrar que el usuario todavía puede salir.
- ✓ Haz que `if (!this.ValidateChildren()) return;` sea la primera línea de `btnSave_Click` y después guarda dentro de `try / catch`. Prueba Save con un campo en el que nunca hayas entrado y escribe la nota del laboratorio.

Entregables requeridos

- ✓ Formulario de alta de clientes con txtName y txtEmail, un botón Save y un botón Cancel
- ✓ Controladores Validating que establecen e.Cancel = true y un mensaje de error para un nombre vacío y un correo electrónico no válido
- ✓ Controlador Validated que recorta el nombre aceptado
- ✓ Controlador de Save que llama a ValidateChildren() y se detiene cuando devuelve false
- ✓ Nota del laboratorio que indica el modo AutoValidate elegido y por qué Cancel sigue funcionando en un formulario no válido

Preguntas de repaso

- ✓ Quitas `e.Cancel = true` de `txtEmail_Validating` pero mantienes la llamada a `SetError`. Describe exactamente qué ve el usuario cuando sale con Tab de «maria.example.com» y después hace clic en Save.
- ✓ ¿Cuáles de tus comprobaciones se ejecutan mientras el usuario edita y cuáles solo porque Save ha llamado a `ValidateChildren()`? Indica una acción del usuario que llegue a Save con un campo cuyo controlador nunca se ha ejecutado.
- ✓ Cambias el formulario de `EnablePreventFocusChange` a `EnableAllowFocusChange`. ¿Qué cambia para el usuario, qué sigue igual y sigue importando `btnCancel.CausesValidation = false`?



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.