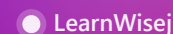


Fondamenti di validazione in Wisej.NET

Guida della lezione · Guida al lab / esame



Guida della lezione

Che cosa significa la validazione in una UI Wisej.NET lato server: il ciclo di vita di Validating e Validated, il rifiuto dell'input non valido con e.Cancel, il comportamento del focus con AutoValidate, CausesValidation sui pulsanti Cancel e ValidateChildren come protezione di Save. Che cosa significa per uno sviluppatore che conosce le basi di C# e sta iniziando a sviluppare form con Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Che cosa significa la validazione in una UI Wisej.NET lato server: il ciclo di vita di Validating e Validated, il rifiuto dell'input non valido con e.Cancel, il comportamento del focus con AutoValidate, CausesValidation sui pulsanti Cancel e ValidateChildren come protezione di Save.

1. Il problema in produzione

ValidationClinic è l'applicazione di esempio che costruisci tu stesso lungo questo corso: la Guida al lab / esame di questo modulo spiega esattamente come iniziare, e l'Esercizio di coding con l'IA può farla assemblare a un assistente. Il suo form di registrazione clienti oggi accetta qualsiasi cosa: un nome vuoto, un'email senza @ o un clic su Save in un form che nessuno ha compilato. La prima idea della maggior parte dei principianti è un'istruzione `if` dentro il pulsante Save, ma così l'utente non riceve alcun feedback mentre lavora e Save diventa un muro di errori alla fine di tutto. Questo modulo mette la validazione dove deve stare in un form Wisej.NET: su ogni editor quando l'utente lo lascia, e un'altra volta quando l'utente fa clic su Save.

2. Concetti chiave

Concetto	Perché è importante
validazione	La decisione se un valore è accettabile prima che l'applicazione lo usi, seguita da un feedback visibile e, quando il valore è sbagliato, dal rifiuto di proseguire.
<code>Validating</code>	L'evento che un controllo genera quando l'utente prova a lasciarlo (o quando il codice chiede la validazione); il suo gestore legge il valore ed è l'unico punto in cui si decide se accettarlo o rifiutarlo.
<code>CancelEventArgs.Cancel</code>	Il flag che un gestore <code>Validating</code> imposta a <code>true</code> per rifiutare il valore: il controllo risulta non valido, il focus può essere trattenuto al suo interno e <code>ValidateChildren()</code> segnala il fallimento.
<code>Validated</code>	L'evento generato solo dopo che <code>Validating</code> è terminato senza annullare; il posto per le rifiniture come togliere gli spazi dal testo, mai per salvare.
<code>AutoValidate</code>	Una proprietà di form, pagina o user control che decide che cosa succede al focus quando un controllo figlio non supera la validazione.
<code>CausesValidation</code>	Un interruttore per singolo controllo; se è impostato a <code>false</code> su un pulsante come Cancel, il clic non valida il campo che l'utente sta lasciando.
<code>ValidateChildren()</code>	Un metodo del contenitore che esegue la validazione dei suoi controlli figli, compresi i campi che l'utente non ha mai visitato, e restituisce <code>false</code> se un gestore ha annullato.

3. Il filo della lezione

La validazione in una UI lato server

In Wisej.NET costruisci un form come qualsiasi classe C#: un `CustomerIntakeForm` con campi chiamati `txtName`, `txtEmail` e `btnSave`. L'oggetto form e i suoi controlli vivono sul server, dentro la sessione dell'utente. Il browser li disegna e riferisce ciò che fa l'utente (digitare, fare clic, spostare il focus), e i tuoi gestori di eventi girano come normale C# sul server. Così, quando l'utente preme Tab per lasciare `txtEmail`, il browser comunica al server che il focus si sta spostando, il server genera l'evento di validazione per `txtEmail`, il tuo gestore legge `txtEmail.Text` e il risultato, per esempio un'icona di errore o il focus trattenuto, viene rimandato indietro e disegnato.

Ne derivano due conseguenze pratiche. La decisione la prende il tuo codice sul server, quindi il browser non può saltarla. E ogni cambio di focus con validazione è un round trip, quindi un gestore deve essere veloce e prevedibile: guarda questo solo campo, poi imposta o cancella un messaggio. Una chiamata a un servizio remoto a ogni pressione di Tab rende il form lento; tieni quel tipo di verifica per Save. Infine, un messaggio di validazione è testo di prodotto scritto per l'utente, come "Inserisci il nome del cliente.", non il messaggio di un'eccezione.

Validating decide, Validated rifinisce

Il ciclo di vita ha un ordine fisso. L'utente modifica un controllo, poi prova a lasciarlo, fa clic su un pulsante che valida o salva il form. Il controllo genera `Validating`. Il tuo gestore esamina il valore: se non è valido, imposta un messaggio e annulla; se è valido, cancella l'eventuale messaggio precedente. Solo quando `Validating` non ha annullato il controllo genera `Validated`.

Tieni separati i ruoli. `Validating` è dove si prende la decisione. `Validated` serve per le rifiniture su un valore che hai già accettato, come togliere gli spazi che l'utente ha digitato attorno a un nome. Non salvare mai da `Validated`: un utente può validare lo stesso campo molte volte prima di fare clic su Save. Puoi collegare entrambi i gestori dall'elenco degli eventi nella finestra Properties di Visual Studio, oppure nel costruttore con `+=`.

```
private void txtName_Validating(object sender, CancelEventArgs e)
{
    if (string.IsNullOrEmpty(txtName.Text))
    {
        e.Cancel = true;
        errorProvider.SetError(txtName, "Inserisci il nome del cliente.");
        return;
    }
    errorProvider.SetError(txtName, "");
}

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

Rifiutare un valore con e.Cancel

Un gestore `Validating` riceve un `CancelEventArgs` come secondo parametro. Impostare `e.Cancel = true` è il rifiuto; il messaggio dell'`ErrorProvider` è la spiegazione, e ti servono entrambi. Un messaggio senza `e.Cancel` è il bug più comune dei principianti: il campo sembra non valido, eppure il valore viene accettato, il focus va avanti e più tardi `ValidateChildren()` segnala che è tutto a posto. Annullare senza un messaggio è il problema opposto: l'utente resta bloccato e non sa perché.

Per il messaggio, aggiungi un `ErrorProvider` al form e imposta il suo `ContainerControl` sul form. `errorProvider.SetError(txtName, "Inserisci il nome del cliente.")` mostra accanto al controllo un'icona di errore con quel testo, e `SetError(txtName, "")` la rimuove. Il ramo che cancella l'errore è importante: senza di esso, un campo corretto continua a mostrare il vecchio errore. Il Modulo 2 approfondisce `ErrorProvider`. Inoltre, non generare eccezioni per errori ordinari come una `@` mancante. Le eccezioni sono per i guasti eccezionali, come un repository che non si riesce a raggiungere.

Scegliere una modalità di AutoValidate

Che cosa succede al focus quando un gestore annulla lo decide la proprietà `AutoValidate` del contenitore: il form, la pagina o lo user control che contiene gli editor. Le scelte sono quattro:

- `EnablePreventFocusChange`: il focus resta nel campo non valido finché non viene corretto.

- `EnableAllowFocusChange` : il focus può spostarsi e l'errore resta visibile; Save continua a rifiutare.
- `Disable` : nulla viene validato al cambio di focus; la validazione avviene solo quando il codice chiama `Validate()` o `ValidateChildren()`.
- `Inherit` : il contenitore usa l'impostazione del suo contenitore padre.

Il form del lab ha due campi obbligatori, quindi `EnablePreventFocusChange` è una scelta chiara e adatta a chi inizia. È anche rigida: se il nome è vuoto e l'utente fa clic su Cancel, il TextBox non valido può rifiutarsi di cedere il focus e Cancel sembra non funzionare. La soluzione è `btnCancel.CausesValidation = false`. Il clic su un controllo con `CausesValidation` a `false` non valida il campo che si sta lasciando, quindi il gestore di Cancel viene eseguito e l'utente può sempre uscire. Save mantiene il valore predefinito `true`. La tua nota del lab registra quale modalità hai scelto e perché Cancel continua a funzionare.

ValidateChildren() come protezione di Save

La validazione dei campi viene eseguita solo per i campi che l'utente lascia davvero. Un utente può digitare un nome e fare clic su Save senza mai entrare in `txtEmail`, quindi il suo gestore `Validating` non è mai stato eseguito. Ecco perché la prima istruzione di `btnSave_Click` è `if (!this.ValidateChildren()) return;`. `ValidateChildren()` avvia la validazione dei controlli figli del form, visitati o no, mostra ogni messaggio impostato dai loro gestori e restituisce `false` se uno di essi ha annullato. Un overload accetta `ValidationConstraints` per limitare quali figli vengono verificati, per esempio solo i controlli abilitati, selezionabili, con tab stop o figli diretti, utile quando parte di un form è nascosta o disabilitata.

`Validate()` è lo strumento più piccolo: valida il controllo che sta perdendo il focus, utile quando `AutoValidate` è `Disable` e vuoi verificare solo il campo corrente. Dopo che `ValidateChildren()` è passato, metti il salvataggio vero e proprio dentro `try / catch`, così una scrittura fallita mostra un messaggio di sistema comprensibile e non viene mai confusa con un errore di input correggibile. La validazione dei campi protegge la modifica; la validazione al salvataggio protegge il salvataggio. Ti servono entrambe.



La regola dei livelli di validazione

Dai un nome al livello prima di scrivere la verifica. La validazione dei campi protegge la modifica e viene eseguita quando l'utente lascia un controllo; la validazione del modello protegge i dati ed esegue le stesse regole dal form, dalla griglia e da qualsiasi servizio; la validazione al salvataggio protegge il database ed esegue di nuovo tutti i livelli prima che venga scritto qualsiasi dato. Mostra ogni errore vicino al suo campo e in un riepilogo visibile, con un linguaggio semplice che dica che cosa fare, e tieni gli errori correggibili dall'utente separati dagli errori di sistema.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Spiegare che cosa significa la validazione in una UI Wisej.NET lato server.
- ✓ Gestire `Validating` e `Validated` sui controlli editor.
- ✓ Rifiutare l'input non valido impostando `CancelEventArgs.Cancel`.
- ✓ Scegliere una modalità di `AutoValidate` per un form o uno user control.

- ✓ Chiamare `Validate` e `ValidateChildren` al momento giusto.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



Lab pratico

Crea il form di registrazione clienti di `ValidationClinic` con i `TextBox txtName` e `txtEmail`, un pulsante `Save` e un pulsante `Cancel`. Collega gestori `Validating` che impostano `e.Cancel = true` e un messaggio dell'`ErrorProvider` quando il nome è vuoto o l'email non contiene `@`, cancellano il messaggio quando il valore è valido, poi toglie gli spazi dal nome nel suo gestore `Validated`. Imposta la modalità `AutoValidate` del form, assicurati che `Cancel` non avvii la validazione così che l'utente possa sempre uscire, e chiama `ValidateChildren()` all'inizio di `btnSave_Click` così che `Save` si fermi quando un campo non è valido. Concludi con una breve nota del lab che spiega il comportamento del focus che hai scelto.

Punto di partenza

ValidationClinic è l'applicazione che costruisci tu stesso, modulo dopo modulo: non c'è nulla da scaricare e non viene installata alcuna soluzione di partenza. Crea in Visual Studio un nuovo progetto `Wisej.NET 4` chiamato `ValidationClinic`, dagli una pagina principale, una griglia dei contatti modificabile e un semplice repository dei contatti in memoria dietro quella griglia, e per ora non validare nulla; ogni modulo successivo estende questo stesso progetto. Due scorciatoie se preferisci non assemblare l'impalcatura a mano: la sezione **Esercizio di coding con l'IA** di questo modulo consegna a ChatGPT o Claude un prompt già pronto che costruisce l'intera soluzione a partire dalla specifica del modulo, e la nostra versione finita di ogni modulo di questo corso si trova nel repository pubblico di esempi `LearnWisej-Samples`, con una cartella per modulo: `Module 1` è quell'impalcatura più la soluzione di questo lab. In questo lab aggiungi il form di registrazione clienti e gli dai i suoi primi due livelli di validazione: la validazione dei campi su ogni editor e la validazione al salvataggio sul pulsante `Save`.

Approccio suggerito

- ✓ Aggiungi un `CustomerIntakeForm` e usa il designer per posizionare `txtName`, `txtEmail`, `btnSave`, `btnCancel` e una label `lblStatus`. Trascina un `ErrorProvider` sul form e imposta il suo `ContainerControl` sul form.
- ✓ Gestisci `txtName.Validating` e `txtEmail.Validating`. In ciascuno verifica il valore; quando è sbagliato, imposta `e.Cancel = true` e chiama `errorProvider.SetError` con un messaggio in linguaggio semplice, e quando è corretto chiama `SetError` con una stringa vuota.
- ✓ Gestisci `txtName.Validated` e toglie lì gli spazi dal testo, così il valore viene rifinito solo dopo essere stato accettato.
- ✓ Imposta `AutoValidate` del form, poi imposta `btnCancel.CausesValidation = false`. Esegui il form, lascia vuoto il nome e fai clic su `Cancel` per dimostrare che l'utente può comunque uscire.
- ✓ Fai di `if (!this.ValidateChildren()) return;` la prima riga di `btnSave_Click`, poi salva dentro `try / catch`. Prova `Save` con un campo in cui non hai mai fatto clic e scrivi la nota del lab.

Deliverable richiesti

- ✓ Form di registrazione clienti con txtName e txtEmail, un pulsante Save e un pulsante Cancel
- ✓ Gestori Validating che impostano e.Cancel = true e un messaggio di errore per un nome vuoto e un'email non valida
- ✓ Gestore Validated che toglie gli spazi dal nome accettato
- ✓ Gestore di Save che chiama ValidateChildren() e si ferma quando restituisce false
- ✓ Nota del lab che indica la modalità AutoValidate scelta e perché Cancel funziona anche su un form non valido

Domande di revisione per lo studente

- ✓ Togli `e.Cancel = true` da `txtEmail_Validating` ma lasci la chiamata a `SetError`. Descrivi con precisione che cosa vede l'utente quando esce con Tab da "maria.example.com" e poi fa clic su Save.
- ✓ Quali delle tue verifiche vengono eseguite mentre l'utente sta modificando, e quali solo perché Save ha chiamato `ValidateChildren()`? Indica un'azione dell'utente che arriva a Save con un campo il cui gestore non è mai stato eseguito.
- ✓ Passi il form da `EnablePreventFocusChange` a `EnableAllowFocusChange`. Che cosa cambia per l'utente, che cosa resta uguale, e `btnCancel.CausesValidation = false` serve ancora?



La regola dei livelli di validazione

Dai un nome al livello prima di scrivere la verifica. La validazione dei campi protegge la modifica e viene eseguita quando l'utente lascia un controllo; la validazione del modello protegge i dati ed esegue le stesse regole dal form, dalla griglia e da qualsiasi servizio; la validazione al salvataggio protegge il database ed esegue di nuovo tutti i livelli prima che venga scritto qualsiasi dato. Mostra ogni errore vicino al suo campo e in un riepilogo visibile, con un linguaggio semplice che dica che cosa fare, e tieni gli errori correggibili dall'utente separati dagli errori di sistema.