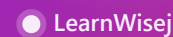


Validazione personalizzata: metodi, servizi e classi ValidationRule

Guida della lezione · Guida al lab / esame



Guida della lezione

Le regole di business che le regole integrate non sanno esprimere: metodi di validazione che restituiscono una stringa, un servizio `ContactValidator` testabile senza la UI, una classe `ValidationRule` personalizzata con `OnValidating`, `OnValidated` e `OnControlCreated`, e regole tra campi che segnalano ogni controllo coinvolto. 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

Le regole di business che le regole integrate non sanno esprimere: metodi di validazione che restituiscono una stringa, un servizio `ContactValidator` testabile senza la UI, una classe `ValidationRule` personalizzata con `OnValidating`, `OnValidated` e `OnControlCreated`, e regole tra campi che segnalano ogni controllo coinvolto.

1. Il problema in produzione

Dopo il Modulo 3 il form di accettazione di `ValidationClinic` sa che un'email ha bisogno di una @ e che l'età deve essere un numero intero. Però salva ancora senza problemi un contatto con la data di inizio successiva alla data di fine, o con una data di nascita che fa avere alla persona undici anni. Le regole integrate verificano la *forma* di un singolo valore; le applicazioni reali hanno bisogno anche di regole sul *significato*, e alcune di queste coinvolgono più di un controllo. Questo modulo dà a quelle regole una collocazione adeguata, così vengono scritte una volta sola, mostrate con chiarezza e mai saltate al clic su Save.

2. Concetti chiave

Concetto	Perché è importante
metodo di validazione personalizzato	Un metodo del form che risponde a una sola domanda e restituisce una stringa vuota quando i dati sono validi, oppure un messaggio in linguaggio semplice quando non lo sono.
servizio di validazione	Una classe come <code>ContactValidator</code> che verifica un modello senza toccare alcun controllo, così le stesse regole vengono eseguite dal form, da una griglia o da uno unit test.
record <code>ValidationMessage</code>	Un piccolo tipo di risultato che contiene il nome di un campo (<code>Field</code>) e un <code>Message</code> ; il servizio ne restituisce una lista invece di impostare direttamente gli errori.
<code>ApplyModelValidation</code>	Il metodo del form che cancella gli errori precedenti, riporta ogni nome di campo al suo controllo con <code>errorProvider.SetError</code> , compila il riepilogo e indica se tutto è passato.
<code>ValidationRule</code>	La classe base di Wisej.NET dietro ogni regola dell'extender Validation; derivando da essa trasformi la tua verifica in una regola che puoi collegare a qualsiasi controllo.
<code>OnValidating</code> / <code>OnValidated</code> / <code>OnControlCreated</code>	I tre override di una classe di regola: verificare il valore e restituire true o false, agire dopo che un valore è passato, e fare la preparazione una volta che il controllo esiste nel browser.
regola tra campi	Una regola la cui risposta dipende da due o più controlli, per esempio la data di inizio prima della data di fine, quindi il suo errore va su ogni controllo coinvolto.

3. Il filo della lezione

Parti da un metodo che restituisce un messaggio

La regola personalizzata più semplice è un normale metodo C#. Legge i valori che gli servono, pone una sola domanda e restituisce una stringa: vuota quando va tutto bene, una frase su cui l'utente può agire quando qualcosa non va. `ValidateStartEndDates` nel lab legge `dtpStart.Value` e `dtpEnd.Value` , restituisce "Entrambe le date sono obbligatorie." quando ne manca una, "La data di inizio deve precedere la data di fine." quando l'ordine è sbagliato, e "" negli altri casi.

Nota che cosa il metodo *non* fa: non tocca l' `ErrorProvider` . Un secondo piccolo metodo prende il testo restituito e lo passa a `errorProvider.SetError` per ciascun selettore. Poiché un messaggio vuoto cancella l'errore, le stesse due righe mostrano il problema e lo rimuovono una volta corretto. Tenere la decisione (la stringa) separata dalla visualizzazione (le chiamate a `SetError`) è l'abitudine che rende facile il passo successivo.

Sposta le regole ripetute in un servizio di validazione

Appena la stessa verifica serve in un secondo punto, per esempio nel form di accettazione e in una finestra di inserimento rapido, un metodo su un solo form non basta più. Il lab sposta le regole su nome, email e data di nascita in una classe `ContactValidator`. Il suo metodo `Validate` riceve un `ContactEditModel`, un semplice oggetto con le proprietà `Name`, `Email` e `BirthDate`, e restituisce una lista di record `ValidationMessage(Field, Message)`: "Il nome è obbligatorio.", "L'email deve contenere @.", "La data di nascita non può essere nel futuro."

Il servizio non sa nulla delle caselle di testo, quindi puoi chiamarlo da un test con un modello costruito a mano e controllare la lista che restituisce, senza browser e senza una sessione Wisej.NET in esecuzione. Al form serve poi un solo adattatore, `ApplyModelValidation`. Chiama `errorProvider.Clear()`, esegue il servizio di validazione, usa uno `switch` per trasformare `"Name"`, `"Email"` e `"BirthDate"` in `txtName`, `txtEmail` e `dtpBirthDate`, imposta ogni errore, unisce tutti i messaggi in `validationSummaryLabel` e restituisce true solo quando la lista è vuota. Un messaggio senza un controllo corrispondente arriva comunque al riepilogo, quindi non si perde nulla. Il Modulo 5 costruirà il modello tramite data binding; qui lo compili tu a partire dai controlli.

Crea una classe derivata da ValidationRule

Alcune regole appartengono a un solo tipo di controllo e dovrebbero spostarsi da un form all'altro come quelle integrate. Per queste, deriva da `ValidationRule`. `MinimumAgeValidationRule` ha una proprietà `MinimumAge` con valore predefinito 18, e il suo costruttore chiama `base("MinimumAge", "La persona è troppo giovane.")`, che dà alla regola il suo nome e il messaggio di non validità che l'extender mostra quando la regola fallisce.

La colleghi esattamente come le regole del Modulo 3, passando un'istanza a `validation.SetValidationRules` per `dtpBirthDate`. Da quel momento l'extender Validation la esegue ogni volta che il selettore viene validato, quando l'utente lascia il campo e quando `ValidateChildren()` viene eseguito al clic su Save, e invia il messaggio allo stesso `ErrorProvider` di tutte le altre regole.

A che cosa servono OnValidating, OnValidated e OnControlCreated

`OnValidating(Control control)` è il cuore della regola. Restituisce `true` quando il valore è valido e `false` quando non lo è; non restituisce un messaggio, perché il messaggio è stato fornito al costruttore. La versione del lab verifica che il controllo sia un `DateTimePicker` il cui `Value` è un `DateTime`, calcola l'età e sottrae un anno se quest'anno il compleanno non c'è ancora stato:

```
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; // niente da verificare: sui valori vuoti decide Required
}
```

Gli altri due override sono facoltativi. `OnValidated` viene eseguito dopo che un valore è passato, quindi è il posto giusto per il lavoro successivo, come formattare il valore accettato. `OnControlCreated` viene eseguito una volta che il controllo è stato creato sul client, quindi qualsiasi preparazione che richiede l'esistenza del widget nel browser va lì e non nel costruttore. Per una regola da principiante è normale fare l'override del solo `OnValidating`.

Regole tra campi che non confondono gli utenti

Una data di inizio successiva alla data di fine non è colpa di uno solo dei due selettori, quindi non darne la colpa a uno solo. Imposta lo stesso messaggio su `dtpStart` e `dtpEnd`, aggiungi una riga al riepilogo e cancella entrambi gli errori appena l'ordine è corretto. Evita `e.Cancel = true` per questo tipo di regola: bloccherebbe il focus nel primo selettore, mentre la correzione potrebbe essere nel secondo.

Poi rendi Save l'ultimo cancello. `btnSave_Click` esegue prima le regole dei campi con `ValidateChildren()`, che include `MinimumAgeValidationRule`. Se passano, costruisce il modello e chiama `ApplyModelValidation`, e solo dopo esegue la verifica delle date, perché altrimenti `errorProvider.Clear()` cancellerebbe gli errori sulle date. Solo quando tutti i livelli passano chiama il repository. Una regola non superata è un normale feedback del prodotto, non un'eccezione, quindi il database non viene mai aperto per dati che sai già essere sbagliati.



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:

- ✓ Scrivere metodi di validazione personalizzati per regole di business di livello base.
- ✓ Spostare le regole ripetute in un servizio di validazione riutilizzabile.

- ✓ Creare una classe personalizzata derivata da `ValidationRule`.
- ✓ Comprendere `OnValidating`, `OnValidated` e `OnControlCreated`.
- ✓ Applicare la validazione tra campi senza confondere gli utenti.

Guida al lab / esame

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

Lab pratico

Aggiungi la validazione personalizzata a ValidationClinic in tre passi. Scrivi `ValidateStartEndDates` come metodo che restituisce una stringa vuota quando i dati sono validi e un messaggio quando la data di inizio è successiva alla data di fine, e imposta quel messaggio sia su `datetimePickerStart` sia su `datetimePickerEnd`. Sposta le regole su nome, email e data di nascita in un servizio `ContactValidator` che restituisce una lista di record `ValidationMessage` e riporta ogni nome di campo al suo controllo in `ApplyModelValidation`. Poi crea `MinimumAgeValidationRule` derivata da `ValidationRule` con una proprietà `MinimumAge`, fai l'override di `OnValidating` per calcolare l'età dal `DateTimePicker`, collegala a `datetimePickerBirthDate` tramite l'extender `Validation` ed esegui il servizio in `btnSave_Click` prima che venga salvato qualsiasi dato.

Punto di partenza

Apri `ValidationClinic` come l'hai lasciato dopo il Modulo 3: il form di accettazione ha un extender `Validation` e un `ErrorProvider` che verificano email, età, telefono, fido e codice cliente, più una `validationSummaryLabel`. I selettori `datetimePickerBirthDate`, `datetimePickerStart` e `datetimePickerEnd` non hanno ancora regole, e `btnSave_Click` salva tutto ciò che `ValidateChildren()` lascia passare.

Approccio suggerito

- ✓ Scrivi `ValidateStartEndDates` in modo che restituisca "" o un messaggio, poi un piccolo helper che passa quella stessa stringa a `errorProvider.SetError` sia per `datetimePickerStart` sia per `datetimePickerEnd`. Prova un ordine sbagliato, poi correggilo, e verifica che entrambe le icone compaiano ed entrambe scompaiano.
- ✓ Crea un record `ValidationMessage(Field, Message)` e una classe `ContactValidator` il cui metodo `Validate` verifica in un `ContactEditModel` che il nome non sia vuoto, che l'email ci sia e contenga la @ e che la data di nascita non sia nel futuro. Non deve fare riferimento ad alcun controllo.
- ✓ Aggiungi `ApplyModelValidation` al form: cancella l' `ErrorProvider`, esegui il servizio di validazione, associa "Name", "Email" e "BirthDate" ai rispettivi controlli, compila la label di riepilogo e restituisci se la lista era vuota.
- ✓ Crea `MinimumAgeValidationRule` derivata da `ValidationRule`, chiama `base("MinimumAge", "La persona è troppo giovane.")`, fai l'override di `OnValidating` per calcolare l'età e collega un'istanza a `datetimePickerBirthDate` con `validation.SetValidationRules`.
- ✓ Riscrivi `btnSave_Click` a livelli: fermati se `ValidateChildren()` fallisce, fermati se `ApplyModelValidation` o la verifica delle date falliscono, e solo allora chiama il repository dentro un `try/catch` per i veri errori di sistema. Aggiungi al file del lab una breve nota che elenca ogni API che hai usato.

Deliverable richiesti

- ✓ Metodo `ValidateStartEndDates` che imposta lo stesso errore su entrambi i selettori di data e li cancella entrambi una volta corretti
- ✓ Servizio `ContactValidator` che restituisce record `ValidationMessage` per nome, email e data di nascita
- ✓ Metodo `ApplyModelValidation` che associa ogni nome di campo al suo controllo e al riepilogo
- ✓ Classe `MinimumAgeValidationRule` con override di `OnValidating`, collegata a `dtpBirthDate`
- ✓ Gestore di `Save` che esegue le regole dei campi, poi il servizio di validazione, e solo allora salva

Domande di revisione per lo studente

- ✓ La verifica dell'età minima potrebbe stare in `ContactValidator` o in `MinimumAgeValidationRule`. Che cosa percepisce di diverso l'utente nei due casi, e quando vorresti averle entrambe?
- ✓ Il tuo gestore di `Save` chiama `ApplyModelValidation` dopo aver impostato gli errori sulle date, e le icone delle date non compaiono mai. Spiega perché, e quale ordine risolve il problema.
- ✓ Come dimostreresti che `ContactValidator` rifiuta un'email senza @ senza avviare l'app né aprire un browser?



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.