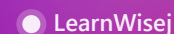


Validazione, ErrorProvider e feedback all'utente

Guida della lezione · Guida al lab / esame



Guida della lezione

Tre livelli di validazione, DataAnnotations sui modelli di modifica, un servizio TicketValidator che restituisce messaggi per campo e di riepilogo, ErrorProvider e riepiloghi come feedback in Wisej.NET, e DbUpdateException come ultimo livello di sicurezza. Che cosa significa per uno sviluppatore Wisej.NET che lavora con database reali — e come affrontare questo modulo.



Focus di produzione

Tre livelli di validazione, DataAnnotations sui modelli di modifica, un servizio TicketValidator che restituisce messaggi per campo e di riepilogo, ErrorProvider e riepiloghi come feedback in Wisej.NET, e DbUpdateException come ultimo livello di sicurezza.

1. Il problema in produzione

L'editor dei ticket del Modulo 4 salva qualsiasi cosa l'utente abbia digitato. Un titolo vuoto, un ticket senza cliente o un ticket chiuso con scadenza la settimana prossima passano tutti per `SaveChangesAsync` e vengono fermati, se va bene, dal database, che risponde con un'eccezione del provider che un agente di supporto non sa leggere. Gli utenti hanno bisogno di messaggi immediati e comprensibili prima che il database rifiuti un record, e l'applicazione ha bisogno di un'ultima linea di difesa che resti al suo posto quando l'interfaccia viene aggirata da un'importazione, da una seconda schermata o da un bug.

2. Concetti chiave

Concetto	Perché è importante
tre livelli di validazione	La validazione dell'interfaccia dà un feedback immediato, la validazione di dominio impone regole che devono valere ovunque e i vincoli del database garantiscono la coerenza finale; ogni livello intercetta ciò che è sfuggito a quello sopra.
DataAnnotations sul modello di modifica	Attributi come <code>[Required]</code> e <code>[StringLength]</code> su <code>TicketEditModel</code> descrivono le regole di input dell'utente in un unico punto dichiarativo, separato dall'entità EF Core.
validazione esplicita	<code>Validator.TryValidateObject</code> esegue le annotation su richiesta; EF Core non le esegue per te prima di <code>SaveChangesAsync</code> , quindi è il form a doverlo chiamare.
TicketValidator e ValidationMessage	Un servizio che restituisce elementi <code>ValidationMessage(FieldName, Message)</code> , così l'insieme di regole è testabile e il form si limita a mappare i messaggi sui controlli.
ErrorProvider	L'extender Wisej.NET che mostra un'icona e un tooltip accanto al controllo non valido tramite <code>SetError</code> , e che può anche essere collegato agli errori di dati dell'elemento corrente.
riepilogo di validazione	Una label o un <code>AlertBox</code> che elenca gli errori tra più campi che non appartengono a un singolo controllo, come un ticket chiuso con una data di scadenza futura.
DbUpdateException	L'eccezione che EF Core solleva quando il database rifiuta la modifica; è l'ultimo livello di sicurezza e va tradotta in un messaggio comprensibile, mai mostrata così com'è.

3. Il filo della lezione

Tre livelli, un'unica autorità finale

Un'applicazione di produzione Wisej.NET ed EF Core valida in tre punti. Il livello dell'interfaccia intercetta gli errori evidenti mentre l'utente è ancora sulla schermata: un titolo mancante, una data di scadenza fuori intervallo, un cliente mai selezionato. Il livello di dominio contiene le regole di business che devono essere vere indipendentemente da quale schermata, importazione o job in background crei il record. Il livello del database, con le sue colonne NOT NULL, gli indici univoci, le chiavi esterne e il token `RowVersion` del Modulo 2, è l'autorità finale per la coerenza.

I livelli non sono alternativi. Il feedback dell'interfaccia è una cortesia che rende veloce il caso comune; il database resta l'unico posto in cui unicità e integrità referenziale possono essere garantite con utenti concorrenti. L'editor di Support Desk quindi valida prima di salvare e si aspetta comunque che ogni tanto il database dica di no.

Annota il modello di modifica, poi validalo esplicitamente

Il Modulo 4 ha introdotto `TicketEditModel` come oggetto a cui si collega l'editor. Questo modulo lo decora con `System.ComponentModel.DataAnnotations`: `[Required, StringLength(180)]` su `Title`, `[StringLength(4000)]` su `Description`, `[Required]` su `Status`, `Priority`, `CustomerId` e `CategoryId`. Le chiavi esterne sono `int?` nullable di proposito: una `ComboBox` senza selezione produce

null, ed è proprio null che `[Required]` rifiuta, mentre un `int` non nullable passerebbe in silenzio come zero.

Le annotation descrivono regole; qualcosa deve eseguirle. `Validator.TryValidateObject` con `validateAllProperties: true` valuta ogni attributo e riempie una lista di elementi `ValidationResult`. Il lab avvolge quella chiamata in un servizio `TicketValidator` che appiattisce i risultati in record `ValidationMessage` e aggiunge le regole che le annotation non possono esprimere. La regola tra campi è l'esempio classico: un ticket chiuso non può avere una data di scadenza futura.

```
public IReadOnlyList<ValidationMessage> Validate(TicketEditModel model)
{
    var results = new List<ValidationResult>();
    var context = new ValidationContext(model);
    Validator.TryValidateObject(model, context, results, validateAllProperties: true);

    var messages = results
        .SelectMany(r => r.MemberNames.DefaultIfEmpty(null),
            (r, member) => new ValidationMessage(member, r.ErrorMessage ?? "Invalid
value."))
        .ToList();

    if (model.Status == "Closed" && model.DueDate?.Date > DateTime.Today)
        messages.Add(new ValidationMessage(nameof(model.DueDate), "Closed tickets cannot
have a future due date."));
    return messages;
}
```

Poiché il validatore è un semplice servizio senza riferimenti a `Wisej.Web`, i cinque casi di test negativi richiesti dal lab girano nel progetto Tests senza un form, una sessione o un database.

Perché gli attributi di mapping non sono una pipeline di validazione

Anche EF Core capisce `[Required]` e `[MaxLength]`, ma su un'entità sono istruzioni di mapping: decidono se una colonna è nullable e quanto è larga. EF Core non esegue un passaggio completo di validazione DataAnnotations prima di `SaveChangesAsync`. Un titolo di 300 caratteri su un'entità mappata a 180 non viene intercettato in .NET; viene inviato al database, che lo tronca o lancia un'eccezione a seconda del provider. Tieni la forma della persistenza nella Fluent API su `SupportDeskContext`, tieni le regole di input dell'utente sul modello di modifica, e chiama tu il validatore.

Il test pratico è semplice: se una regola ha bisogno di un messaggio che un essere umano deve leggere, appartiene al modello di modifica e a `TicketValidator`. Se è un fatto che riguarda la colonna, appartiene alla configurazione del modello e alla migrazione.

ErrorProvider per i campi, un riepilogo per le regole tra campi

L' `ErrorProvider` di Wisej.NET associa a qualsiasi controllo un'icona rossa con un tooltip. L'editor tiene una sola istanza, creata sul form, e segue una sequenza fissa in `SaveAsync`: chiama `Clear()` così spariscono le icone dell'esecuzione precedente, chiama `EndEdit` sulla `BindingSource` così il modello contiene ciò che l'utente ha digitato, esegue `TicketValidator.Validate` e poi scorre i messaggi. Un messaggio il cui `FieldName` è `Title`, `CustomerId`, `CategoryId` o `DueDate` diventa

`errorProvider.SetError(control, message)` sulla `TextBox`, `ComboBox` o `DateTimePicker`

corrispondente. Il provider può anche essere collegato all'elemento corrente di una sorgente dati, così un modello che segnala i propri errori illumina i controlli automaticamente, e l'extender Validation copre le regole semplici per singolo controllo senza codice.

Le regole tra campi non hanno una sede unica, quindi ogni messaggio viene anche unito in una label di riepilogo, oppure mostrato in un `AlertBox` quando il form è affollato. Save resta disabilitato finché il modello non è valido e viene riabilitato nel `finally`, e il metodo ritorna prima che venga mai creato un `DbContext`. Se la validazione fallisce, EF Core non viene chiamato.

DbUpdateException come ultimo livello di sicurezza

Anche un modello completamente validato può essere rifiutato. L'esempio del lab è un numero di ticket duplicato: due agenti creano ticket nello stesso secondo, l'indice univoco su `Number` scatta al secondo salvataggio ed EF Core lancia `DbUpdateException`. La stessa eccezione copre le violazioni di chiave esterna quando una categoria è stata eliminata mentre l'editor era aperto, e gli errori su colonne obbligatorie di cui l'interfaccia non sapeva nulla.

Il gestore fa tre cose. Registra sul server l'eccezione completa, con l'errore interno del provider, tramite il logger dell'applicazione. Mostra all'utente una sola frase semplice, come "Non è stato possibile salvare il ticket perché il database ha rifiutato la modifica. Verifica i campi obbligatori e i valori dei lookup." E quando un lookup potrebbe essere obsoleto, ricarica le liste di lookup così l'utente può scegliere un valore valido. Non mostra mai l'istruzione SQL, il nome del vincolo o qualsiasi cosa possa contenere una stringa di connessione. `DbUpdateConcurrencyException` deriva da `DbUpdateException`, quindi intercettala per prima; il Modulo 7 le dà un dialogo tutto suo.



Errori tipici

Dare per scontato che EF Core esegua automaticamente tutta la validazione DataAnnotations prima di `SaveChanges`; si limita a mapparle. Mostrare agli utenti SQL grezzo o errori del provider, che fanno trapelare dettagli dello schema e a volte informazioni sulla connessione. Mettere regole di business riutilizzabili solo nei gestori dei pulsanti, dove non si possono testare e mancano in silenzio nella prossima schermata che salva un ticket.



La regola del data binding

Tieni separati i due cicli di vita. Form Wisej.NET, BindingSource e riga selezionata vivono per la sessione; un DbContext vive per un'operazione. Crealo dalla factory, interroga o salva, eliminalo, e non lasciarlo mai diventare stato di sessione.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Usare `System.ComponentModel.DataAnnotations` o validatori personalizzati sui modelli di modifica.

- ✓ Mostrare gli errori di validazione con `ErrorProvider`, l'extender `Validation`, label e messaggi di riepilogo.
- ✓ Capire che gli attributi di mapping di EF Core non sono un sistema completo di validazione automatica dell'interfaccia.
- ✓ Gestire i vincoli del database e `DbUpdateException` come ultimo livello di sicurezza.

Guida al lab / esame

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



Lab pratico

Aggiungi la validazione all'editor dei ticket di Support Desk: DataAnnotations su TicketEditModel, un TicketValidator che restituisce errori per campo e di riepilogo, un ErrorProvider azzerato prima di ogni esecuzione e impostato per Title, Customer, Category e DueDate, una label di riepilogo per le regole tra più campi, Save disabilitato finché il modello non è valido, un gestore di DbUpdateException che mostra un messaggio comprensibile per un numero di ticket duplicato, e almeno cinque casi di test negativi.

Punto di partenza

Apri la soluzione `SupportDesk` del Modulo 4: il browser dei ticket, i form modali Add Ticket ed Edit Ticket, `TicketEditModel` e un `TicketEditorForm` con una `BindingSource` e un `ErrorProvider` esistono già, e `SaveAsync` contiene ancora il segnaposto di validazione che lascia passare qualsiasi modello fino a `SaveChangesAsync`. Il progetto Services non ha ancora un validatore e il progetto Tests è vuoto.

Approccio suggerito

- ✓ Decora `TicketEditModel` nel progetto Services con `[Required, StringLength(180)]` su `Title`, `[StringLength(4000)]` su `Description` e `[Required]` su `Status`, `Priority`, `CustomerId` e `CategoryId`; tieni le chiavi esterne nullable così un lookup non selezionato viene intercettato come null.
- ✓ Aggiungi un record `ValidationMessage(FieldName, Message)` e un servizio `TicketValidator` il cui metodo `Validate` esegue `Validator.TryValidateObject` con `validateAllProperties: true`, appiattisce i risultati per nome del membro e aggiunge la regola tra campi secondo cui un ticket chiuso non può avere una data di scadenza futura, con un `FieldName` null oppure `DueDate` come campo; registralo nella DI di Microsoft accanto alla factory del Context.
- ✓ In `SaveAsync` chiama `errorProvider.Clear()` ed `editBindingSource.EndEdit()`, poi chiama il validatore; scrivi un metodo `ShowValidation` che mappa `Title`, `CustomerId`, `CategoryId` e `DueDate` su `SetError` per la `TextBox`, la `ComboBox` e il `DateTimePicker` corrispondenti, unisce tutti i messaggi nella label di riepilogo e ritorna prima che venga creato qualsiasi `DbContext`.
- ✓ Tieni Save disabilitato finché il modello non è valido e mentre un salvataggio è in corso, ripristinandolo nel `finally`; poi aggiungi sotto il catch di concorrenza un `catch (DbUpdateException)` che registra l'eccezione completa lato server, mostra una sola frase comprensibile tramite `AlertBox` e ricarica le liste di lookup quando una chiave esterna potrebbe essere obsoleta.
- ✓ Nel progetto Tests scrivi almeno cinque casi negativi contro il solo `TicketValidator` (titolo vuoto, titolo oltre i 180 caratteri, cliente mancante, categoria mancante, ticket chiuso con data di scadenza futura), poi riproduci il numero di ticket duplicato contro l'indice univoco e verifica che il messaggio non contenga SQL.

Deliverable richiesti

- ✓ DataAnnotations su TicketEditModel e un TicketValidator che restituisce errori per campo e di riepilogo
- ✓ ErrorProvider azzerato prima di ogni esecuzione e impostato per Title, Customer, Category e DueDate
- ✓ Label di riepilogo per le regole tra più campi e Save disabilitato finché il modello non è valido
- ✓ DbUpdateException gestita con un messaggio comprensibile per un numero di ticket duplicato
- ✓ Almeno cinque casi di test negativi

Domande di revisione per lo studente

- ✓ Se togliessi tutte le DataAnnotations da `TicketEditModel` ma le lasciassi sull'entità `Ticket`, quali dei tuoi cinque test negativi fallirebbero ancora prima di `SaveChangesAsync`, e perché?
- ✓ La regola sui ticket chiusi riporta `DueDate` come campo. Che cosa cambierebbe per l'utente se non riportasse alcun campo, e quando sarebbe la scelta migliore?
- ✓ Due agenti creano un ticket nello stesso secondo e il secondo salvataggio lancia `DbUpdateException`. Che cosa vede il secondo agente, che cosa registra il server, e quale dei due mostreresti a un cliente?



La regola del data binding

Tieni separati i due cicli di vita. Form Wisej.NET, BindingSource e riga selezionata vivono per la sessione; un DbContext vive per un'operazione. Crealo dalla factory, interroga o salva, eliminalo, e non lasciarlo mai diventare stato di sessione.