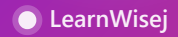


Eigene Validierung: Methoden, Services und ValidationRule-Klassen

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Geschäftsregeln, die die integrierten Regeln nicht ausdrücken können: Validierungsmethoden, die einen String zurückgeben, ein ContactValidator-Service, der sich ohne UI testen lässt, eine eigene ValidationRule-Klasse mit OnValidating, OnValidated und OnControlCreated sowie feldübergreifende Regeln, die jedes beteiligte Control markieren. Was das für Entwickler bedeutet, die grundlegendes C# beherrschen und mit der Formularentwicklung in Wisej.NET beginnen – und wie Sie dieses Modul am besten durcharbeiten.



Fokus: Produktivbetrieb

Geschäftsregeln, die die integrierten Regeln nicht ausdrücken können: Validierungsmethoden, die einen String zurückgeben, ein ContactValidator-Service, der sich ohne UI testen lässt, eine eigene ValidationRule-Klasse mit OnValidating, OnValidated und OnControlCreated sowie feldübergreifende Regeln, die jedes beteiligte Control markieren.

1. Das Problem im Produktivbetrieb

Nach Modul 3 weiß das Aufnahmeformular von ValidationClinic, dass eine E-Mail-Adresse ein @ braucht und ein Alter eine ganze Zahl sein muss. Trotzdem speichert es anstandslos einen Kontakt, dessen Startdatum nach dem Enddatum liegt, oder ein Geburtsdatum, nach dem die Person elf Jahre alt ist. Die integrierten Regeln prüfen die *Form* eines Werts; echte Anwendungen brauchen aber auch Regeln zur *Bedeutung*, und manche dieser Regeln betreffen mehr als ein Control. Dieses Modul gibt solchen Regeln ein ordentliches Zuhause, damit sie einmal geschrieben, klar angezeigt und beim Speichern nie übersprungen werden.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
eigene Validierungsmethode	Eine Methode im Formular, die genau eine Frage beantwortet und einen leeren String zurückgibt, wenn die Daten gültig sind, oder eine verständliche Meldung, wenn nicht.
Validierungs-Service	Eine Klasse wie <code>ContactValidator</code> , die ein Modell prüft, ohne ein Control anzufassen, sodass dieselben Regeln aus dem Formular, einem Grid oder einem Unit-Test laufen.
<code>ValidationMessage</code> -Record	Ein winziger Ergebnistyp mit einem <code>Field</code> -Namen und einer <code>Message</code> ; der Service gibt eine Liste davon zurück, statt selbst Fehler zu setzen.
<code>ApplyModelValidation</code>	Die Formularmethode, die alte Fehler entfernt, jeden Feldnamen mit <code>errorProvider.SetError</code> wieder seinem Control zuordnet, die Zusammenfassung füllt und meldet, ob alles bestanden wurde.
<code>ValidationRule</code>	Die Wisej.NET-Basisklasse hinter jeder Regel des Validation-Extenders; leiten Sie davon ab, wird Ihre eigene Prüfung zu einer Regel, die Sie an jedes Control hängen können.
<code>OnValidating</code> / <code>OnValidated</code> / <code>OnControlCreated</code>	Die drei Überschreibungen einer Regelklasse: den Wert prüfen und true oder false zurückgeben, handeln, nachdem ein Wert bestanden hat, und Vorbereitungen treffen, sobald das Control im Browser existiert.
feldübergreifende Regel	Eine Regel, deren Antwort von zwei oder mehr Controls abhängt, etwa Startdatum vor Enddatum, sodass ihr Fehler an jedes beteiligte Control gehört.

3. Der rote Faden der Lektion

Mit einer Methode beginnen, die eine Meldung zurückgibt

Die einfachste eigene Regel ist eine ganz normale C#-Methode. Sie liest die Werte, die sie braucht, stellt eine Frage und gibt einen String zurück: leer, wenn alles in Ordnung ist, und einen Satz, nach dem der Benutzer handeln kann, wenn nicht. `ValidateStartEndDates` im Lab liest `dtoStart.Value` und `dtoEnd.Value`, gibt „Both dates are required.“ zurück, wenn eines davon fehlt, „Start date must be before end date.“, wenn die Reihenfolge falsch ist, und sonst `""`.

Beachten Sie, was die Methode *nicht* tut: Sie fasst den `ErrorProvider` nicht an. Eine zweite kleine Methode nimmt den zurückgegebenen Text und übergibt ihn für jede Datumsauswahl an `errorProvider.SetError`. Weil eine leere Meldung einen Fehler entfernt, zeigen dieselben zwei Zeilen das Problem an und entfernen es wieder, sobald es behoben ist. Die Entscheidung (den String) von der Anzeige (den `SetError` -Aufrufen) getrennt zu halten, ist die Gewohnheit, die den nächsten Schritt leicht macht.

Wiederholte Regeln in einen Validierungs-Service verschieben

Sobald dieselbe Prüfung an einer zweiten Stelle gebraucht wird, zum Beispiel im Aufnahmeformular und in einem Dialog zum schnellen Anlegen, reicht eine Methode in einem Formular nicht mehr. Das Lab verschiebt die Regeln für Name, E-Mail und Geburtsdatum in eine Klasse `ContactValidator`. Ihre Methode `Validate` erhält ein `ContactEditModel`, ein einfaches Objekt mit den Eigenschaften `Name`, `Email` und `BirthDate`, und gibt eine Liste von `ValidationMessage(Field, Message)`-Records zurück: „Name is required.“, „Email must contain @.“, „Birth date cannot be in the future.“.

Der Service weiß nichts über Textfelder, daher können Sie ihn aus einem Test mit einem selbst gebauten Modell aufrufen und die zurückgegebene Liste prüfen, ganz ohne Browser und ohne laufende Wisej.NET-Session. Das Formular braucht dann nur noch einen Adapter, `ApplyModelValidation`. Er ruft `errorProvider.Clear()` auf, führt den Validator aus, macht mit einem `switch` aus `"Name"`, `"Email"` und `"BirthDate"` die Controls `txtName`, `txtEmail` und `dtpBirthDate`, setzt jeden Fehler, fügt alle Meldungen in `validationSummaryLabel` zusammen und gibt nur dann `true` zurück, wenn die Liste leer ist. Eine Meldung ohne passendes Control landet trotzdem in der Zusammenfassung, es geht also nichts verloren. Modul 5 baut das Modell über Data Binding auf; hier füllen Sie es selbst aus den Controls.

Eine von ValidationRule abgeleitete Klasse erstellen

Manche Regeln gehören zu einer bestimmten Art von Control und sollen wie die integrierten Regeln von Formular zu Formular mitwandern. Für diese leiten Sie von `ValidationRule` ab.

`MinimumAgeValidationRule` hat eine Eigenschaft `MinimumAge` mit dem Standardwert 18, und ihr Konstruktor ruft `base("MinimumAge", "The person is too young.")` auf; das gibt der Regel ihren Namen und die Ungültig-Meldung, die der Extender anzeigt, wenn die Regel fehlschlägt.

Sie hängen sie genauso an wie die Regeln aus Modul 3, indem Sie eine Instanz an `validation.SetValidationRules` übergeben, und zwar für `dtpBirthDate`. Ab dann führt der Validation-Extender sie bei jeder Validierung der Datumsauswahl aus, wenn der Benutzer das Feld verlässt und wenn beim Speichern `ValidateChildren()` läuft, und leitet die Meldung an denselben `ErrorProvider` weiter wie jede andere Regel.

Wofür OnValidating, OnValidated und OnControlCreated da sind

`OnValidating(Control control)` ist das Herzstück der Regel. Die Methode gibt `true` zurück, wenn der Wert gültig ist, und `false`, wenn nicht; eine Meldung gibt sie nicht zurück, denn die wurde bereits dem Konstruktor übergeben. Die Lab-Version prüft, ob das Control ein `DateTimePicker` ist, dessen `Value` ein `DateTime` ist, berechnet das Alter und zieht ein Jahr ab, wenn der Geburtstag in diesem Jahr noch nicht war:

```

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;    // nichts zu prüfen: über leere Werte entscheidet Required
}

```

Die beiden anderen Überschreibungen sind optional. `OnValidated` läuft, nachdem ein Wert bestanden hat, und ist damit der Ort für Folgearbeiten wie das Formatieren des akzeptierten Werts.

`OnControlCreated` läuft, sobald das Control auf dem Client erzeugt wurde; jede Einrichtung, für die das Browser-Widget bereits existieren muss, gehört also dorthin und nicht in den Konstruktor. Für eine Einsteigerregel ist es ganz normal, nur `OnValidating` zu überschreiben.

Feldübergreifende Regeln, die Benutzer nicht verwirren

Ein Startdatum nach dem Enddatum ist nicht die Schuld einer einzelnen Datumsauswahl, also schieben Sie sie auch keiner davon allein zu. Setzen Sie dieselbe Meldung auf `dtpStart` und `dtpEnd`, fügen Sie der Zusammenfassung eine Zeile hinzu und entfernen Sie beide Fehler, sobald die Reihenfolge stimmt.

Vermeiden Sie bei dieser Art von Regel `e.Cancel = true`: Es würde den Fokus in der ersten Datumsauswahl festhalten, obwohl die Korrektur vielleicht in der zweiten liegt.

Machen Sie dann das Speichern zur letzten Schranke. `btnSave_Click` führt zuerst mit `ValidateChildren()` die Feldregeln aus, wozu auch `MinimumAgeValidationRule` gehört. Besteht das, baut der Handler das Modell auf und ruft `ApplyModelValidation` auf und führt erst danach die Datumsprüfung aus, weil `errorProvider.Clear()` die Datumsfehler sonst wieder löschen würde. Nur wenn jede Ebene besteht, ruft er das Repository auf. Eine fehlgeschlagene Regel ist normales Produkt-Feedback und keine Exception, daher wird die Datenbank nie für Daten geöffnet, von denen Sie bereits wissen, dass sie falsch sind.



Gewohnheit: Validierungsebenen

Benennen Sie die Ebene, bevor Sie die Prüfung schreiben. Die Feldvalidierung schützt das Bearbeiten und läuft, wenn der Benutzer ein Control verlässt; die Modellvalidierung schützt die Daten und führt dieselben Regeln aus dem Formular, dem Grid und jedem Service aus; die Validierung beim Speichern schützt die Datenbank und führt jede Ebene noch einmal aus, bevor irgendetwas geschrieben wird. Zeigen Sie jeden Fehler nahe an seinem Feld und in einer sichtbaren Zusammenfassung, in verständlicher Sprache, die sagt, was als Nächstes zu tun ist, und halten Sie Fehler, die der Benutzer selbst korrigieren kann, getrennt von Systemfehlern.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ eigene Validierungsmethoden für einfache Geschäftsregeln schreiben.
- ✓ wiederholte Regeln in einen wiederverwendbaren Validierungs-Service verschieben.
- ✓ eine eigene, von `ValidationRule` abgeleitete Klasse erstellen.
- ✓ `OnValidating`, `OnValidated` und `OnControlCreated` verstehen.
- ✓ feldübergreifende Validierung anwenden, ohne Benutzer zu verwirren.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praxis-Lab

Ergänzen Sie ValidationClinic in drei Schritten um eigene Validierung. Schreiben Sie `ValidateStartEndDates` als Methode, die bei gültigen Werten einen leeren String und eine Meldung zurückgibt, wenn das Startdatum nach dem Enddatum liegt, und setzen Sie diese Meldung sowohl auf `ctpStart` als auch auf `ctpEnd`. Verschieben Sie die Regeln für Name, E-Mail und Geburtsdatum in einen `ContactValidator-Service`, der eine Liste von `ValidationMessage-Records` zurückgibt, und ordnen Sie in `ApplyModelValidation` jeden Feldnamen wieder seinem Control zu. Legen Sie dann `MinimumAgeValidationRule` als Ableitung von `ValidationRule` mit einer Eigenschaft `MinimumAge` an, überschreiben Sie `OnValidating`, um das Alter aus dem `DateTimePicker` zu berechnen, hängen Sie die Regel über den `Validation-Extender` an `ctpBirthDate` und führen Sie den Service in `btnSave_Click` aus, bevor irgendetwas gespeichert wird.

Ausgangspunkt

Öffnen Sie ValidationClinic so, wie Sie es nach Modul 3 verlassen haben: Das Aufnahmeformular hat einen `Validation-Extender` und einen `ErrorProvider`, die E-Mail, Alter, Telefon, Kreditlimit und Kundennummer prüfen, dazu ein `validationSummaryLabel`. Die Datumsauswahlen `ctpBirthDate`, `ctpStart` und `ctpEnd` haben noch keine Regeln, und `btnSave_Click` speichert alles, was `ValidateChildren()` durchlässt.

Vorgeschlagenes Vorgehen

- ✓ Schreiben Sie `ValidateStartEndDates` so, dass die Methode `""` oder eine Meldung zurückgibt, und dann eine kleine Hilfsmethode, die genau diesen String an `errorProvider.SetError` übergibt, und zwar für `ctpStart` und `ctpEnd`. Probieren Sie eine falsche Reihenfolge aus, korrigieren Sie sie dann und prüfen Sie, dass beide Icons erscheinen und beide wieder verschwinden.
- ✓ Legen Sie einen Record `ValidationMessage(Field, Message)` und eine Klasse `ContactValidator` an, deren Methode `Validate` ein `ContactEditModel` auf einen leeren Namen, eine fehlende E-Mail-Adresse oder eine ohne @ und ein Geburtsdatum in der Zukunft prüft. Sie darf kein Control referenzieren.
- ✓ Ergänzen Sie das Formular um `ApplyModelValidation`: den `ErrorProvider` leeren, den Validator ausführen, `"Name"`, `"Email"` und `"BirthDate"` ihren Controls zuordnen, das Zusammenfassungs-Label füllen und zurückgeben, ob die Liste leer war.
- ✓ Legen Sie `MinimumAgeValidationRule` als Ableitung von `ValidationRule` an, rufen Sie `base("MinimumAge", "The person is too young.")` auf, überschreiben Sie `OnValidating`, um das Alter zu berechnen, und hängen Sie eine Instanz an `ctpBirthDate`; verwenden Sie dazu `validation.SetValidationRules`.
- ✓ Schreiben Sie `btnSave_Click` in Ebenen neu: abbrechen, wenn `ValidateChildren()` fehlschlägt, abbrechen, wenn `ApplyModelValidation` oder die Datumsprüfung fehlschlägt, und

erst dann das Repository innerhalb eines `try/catch` für echte Systemfehler aufrufen. Ergänzen Sie Ihre Lab-Datei um eine kurze Notiz, die jede verwendete API nennt.

Geforderte Ergebnisse

- ✓ Methode `ValidateStartEndDates`, die denselben Fehler auf beide Datumsauswahlen setzt und bei der Korrektur beide zurücksetzt
- ✓ `ContactValidator-Service`, der `ValidationMessage-Records` für Name, E-Mail und Geburtsdatum liefert
- ✓ Methode `ApplyModelValidation`, die jeden Feldnamen seinem Control und der Zusammenfassung zuordnet
- ✓ Klasse `MinimumAgeValidationRule`, die `OnValidating` überschreibt und an `dateTimePicker1` hängt
- ✓ Speichern-Handler, der zuerst die Feldregeln, dann den Validator-Service ausführt und erst danach speichert

Wiederholungsfragen

- ✓ Die Prüfung auf das Mindestalter könnte in `ContactValidator` oder in `MinimumAgeValidationRule` stehen. Was erlebt der Benutzer jeweils anders, und wann würden Sie beides wollen?
- ✓ Ihr Speichern-Handler ruft `ApplyModelValidation` auf, nachdem er die Datumsfehler gesetzt hat, und die Datums-Icons erscheinen nie. Erklären Sie, warum, und welche Reihenfolge das behebt.
- ✓ Wie würden Sie nachweisen, dass `ContactValidator` eine E-Mail-Adresse ohne @ ablehnt, ohne die App zu starten oder einen Browser zu öffnen?



Gewohnheit: Validierungsebenen

Benennen Sie die Ebene, bevor Sie die Prüfung schreiben. Die Feldvalidierung schützt das Bearbeiten und läuft, wenn der Benutzer ein Control verlässt; die Modellvalidierung schützt die Daten und führt dieselben Regeln aus dem Formular, dem Grid und jedem Service aus; die Validierung beim Speichern schützt die Datenbank und führt jede Ebene noch einmal aus, bevor irgendetwas geschrieben wird. Zeigen Sie jeden Fehler nahe an seinem Feld und in einer sichtbaren Zusammenfassung, in verständlicher Sprache, die sagt, was als Nächstes zu tun ist, und halten Sie Fehler, die der Benutzer selbst korrigieren kann, getrennt von Systemfehlern.