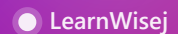


Grundlagen der Validierung in Wisej.NET

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Was Validierung in einer serverseitigen Wisej.NET-UI bedeutet: der Lebenszyklus aus Validating und Validated, das Abweisen ungültiger Eingaben mit e.Cancel, das Fokusverhalten von AutoValidate, CausesValidation auf Abbrechen-Buttons und ValidateChildren als Schutz vor dem Speichern. 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

Was Validierung in einer serverseitigen Wisej.NET-UI bedeutet: der Lebenszyklus aus Validating und Validated, das Abweisen ungültiger Eingaben mit e.Cancel, das Fokusverhalten von AutoValidate, CausesValidation auf Abbrechen-Buttons und ValidateChildren als Schutz vor dem Speichern.

1. Das Problem im Produktivbetrieb

ValidationClinic ist die Beispielanwendung, die Sie im Laufe dieses Kurses selbst erstellen — der Lab- und Prüfungsleitfaden dieses Moduls beschreibt genau, wie Sie beginnen, und die KI-Coding-Übung lässt sie von einem Assistenten für Sie zusammenbauen. Das Formular zur Kundenaufnahme akzeptiert derzeit alles: einen leeren Namen, eine E-Mail-Adresse ohne @ oder einen Klick auf Speichern in einem Formular, das niemand ausgefüllt hat. Die erste Idee der meisten Einsteiger ist eine `if`-Anweisung im Speichern-Button, aber damit erhält der Benutzer während der Arbeit keinerlei Feedback, und Speichern wird ganz am Ende zu einer Wand aus Fehlermeldungen. Dieses Modul bringt die Validierung dorthin, wo sie in einem Wisej.NET-Formular hingehört: an jeden Editor, wenn der Benutzer ihn verlässt, und noch einmal, wenn der Benutzer auf Speichern klickt.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Validierung	Die Entscheidung, ob ein Wert akzeptabel ist, bevor die Anwendung ihn verwendet, gefolgt von sichtbarem Feedback und, wenn der Wert falsch ist, der Weigerung, fortzufahren.
<code>Validating</code>	Das Event, das ein Control auslöst, wenn der Benutzer es verlassen will (oder wenn Code eine Validierung anfordert); sein Handler liest den Wert und ist die einzige Stelle, an der über Annehmen oder Ablehnen entschieden wird.
<code>CancelEventArgs.Cancel</code>	Das Flag, das ein <code>Validating</code> -Handler auf <code>true</code> setzt, um den Wert abzulehnen: Das Control gilt als ungültig, der Fokus kann darin gehalten werden, und <code>ValidateChildren()</code> meldet einen Fehlschlag.
<code>Validated</code>	Das Event, das erst ausgelöst wird, nachdem <code>Validating</code> ohne Abbruch beendet wurde; der Ort für Aufräumarbeiten wie das Trimmen von Text, niemals für das Speichern.
<code>AutoValidate</code>	Eine Eigenschaft eines Formulars, einer Page oder eines User-Controls, die festlegt, was mit dem Fokus passiert, wenn ein untergeordnetes Control die Validierung nicht besteht.
<code>CausesValidation</code>	Ein Schalter pro Control; steht er bei einem Button wie Abbrechen auf <code>false</code> , validiert ein Klick darauf das Feld nicht, das der Benutzer gerade verlässt.
<code>ValidateChildren()</code>	Eine Methode eines Containers, die die Validierung für seine untergeordneten Controls ausführt, auch für Felder, die der Benutzer nie besucht hat, und <code>false</code> zurückgibt, wenn irgendein Handler abgebrochen hat.

3. Der rote Faden der Lektion

Validierung in einer serverseitigen UI

In Wisej.NET bauen Sie ein Formular wie jede andere C#-Klasse: ein `CustomerIntakeForm` mit Feldern namens `txtName`, `txtEmail` und `btnSave`. Dieses Formularobjekt und seine Controls leben auf dem Server, innerhalb der Session des Benutzers. Der Browser zeichnet sie und meldet, was der Benutzer tut (tippen, klicken, den Fokus wechseln), und Ihre Event-Handler laufen als ganz normales C# auf dem Server. Wenn der Benutzer also mit Tab `txtEmail` verlässt, teilt der Browser dem Server mit, dass sich der Fokus bewegt, der Server löst das Validierungs-Event für `txtEmail` aus, Ihr Handler liest `txtEmail.Text`, und das Ergebnis, etwa ein Fehler-Icon oder ein festgehaltener Fokus, wird zurückgeschickt und gezeichnet.

Das hat zwei praktische Folgen. Die Entscheidung trifft Ihr Servercode, der Browser kann sie also nicht überspringen. Und jeder validierende Fokuswechsel ist ein Roundtrip, deshalb sollte ein Handler schnell und vorhersehbar sein: genau dieses eine Feld ansehen, dann eine Meldung setzen oder entfernen. Eine Abfrage bei einem entfernten Service bei jedem Druck auf Tab lässt das Formular träge wirken; heben Sie solche Prüfungen für das Speichern auf. Und schließlich ist eine Validierungsmeldung Produkttext, der für den Benutzer geschrieben ist, etwa „Enter the customer's name.“, keine Exception-Meldung.

Validating entscheidet, Validated räumt auf

Der Lebenszyklus hat eine feste Reihenfolge. Der Benutzer bearbeitet ein Control und versucht dann, es zu verlassen, einen validierenden Button zu klicken oder das Formular zu speichern. Das Control löst `Validating` aus. Ihr Handler prüft den Wert: Ist er ungültig, setzt er eine Meldung und bricht ab; ist er gültig, entfernt er eine eventuell vorhandene alte Meldung. Nur wenn `Validating` nicht abgebrochen hat, löst das Control `Validated` aus.

Halten Sie die Rollen auseinander. In `Validating` fällt die Entscheidung. `Validated` dient dem Aufräumen eines Werts, den Sie bereits akzeptiert haben, etwa dem Entfernen der Leerzeichen, die ein Benutzer um einen Namen herum getippt hat. Speichern Sie niemals aus `Validated` heraus: Ein Benutzer kann dasselbe Feld viele Male validieren, bevor er auf Speichern klickt. Beide Handler können Sie über die Event-Liste im Eigenschaftfenster von Visual Studio anhängen oder im Konstruktor mit `+=`.

```
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();
}
```

Einen Wert mit e.Cancel ablehnen

Ein `Validating`-Handler erhält als zweiten Parameter ein `CancelEventArgs`. `e.Cancel = true` zu setzen ist die Ablehnung; die `ErrorProvider`-Meldung ist die Erklärung, und Sie brauchen beides. Eine Meldung ohne `e.Cancel` ist der häufigste Einsteigerfehler: Das Feld sieht ungültig aus, trotzdem wird der Wert akzeptiert, der Fokus wandert weiter, und `ValidateChildren()` meldet später, alles sei in Ordnung. Abbrechen ohne Meldung ist das umgekehrte Problem: Der Benutzer steckt fest und weiß nicht, warum.

Für die Meldung fügen Sie dem Formular einen `ErrorProvider` hinzu und setzen seinen `ContainerControl` auf das Formular. `errorProvider.SetError(txtName, "Enter the customer's name.")` zeigt neben dem Control ein Fehler-Icon mit diesem Text an, und `SetError(txtName, "")` entfernt es wieder. Der Zweig zum Entfernen ist wichtig: Ohne ihn zeigt ein korrigiertes Feld weiterhin seinen alten Fehler. Modul 2 behandelt `ErrorProvider` ausführlich. Werfen Sie außerdem keine Exceptions für gewöhnliche Fehler wie ein fehlendes `@`. Exceptions sind für außergewöhnliche Störungen gedacht, etwa ein Repository, das nicht erreichbar ist.

Einen AutoValidate-Modus wählen

Was mit dem Fokus passiert, wenn ein Handler abbricht, legt die Eigenschaft `AutoValidate` des Containers fest: des Formulars, der Page oder des User-Controls, das die Editoren enthält. Es gibt vier

Möglichkeiten:

- `EnablePreventFocusChange` : Der Fokus bleibt im ungültigen Feld, bis es korrigiert ist.
- `EnableAllowFocusChange` : Der Fokus darf wechseln, und der Fehler bleibt sichtbar; Speichern verweigert trotzdem.
- `Disable` : Beim Fokuswechsel wird nichts validiert; die Validierung läuft nur, wenn Code `Validate()` oder `ValidateChildren()` aufruft.
- `Inherit` : Der Container übernimmt die Einstellung seines übergeordneten Elements.

Das Lab-Formular hat zwei Pflichtfelder, daher ist `EnablePreventFocusChange` eine klare, einsteigerfreundliche Wahl. Sie ist aber auch streng: Ist der Name leer und klickt der Benutzer auf Abbrechen, gibt die ungültige TextBox den Fokus womöglich nicht ab, und Abbrechen scheint kaputt zu sein. Die Lösung ist `btnCancel.CausesValidation = false`. Ein Klick auf ein Control, dessen `CausesValidation` auf `false` steht, validiert das verlassene Feld nicht, also läuft der Abbrechen-Handler, und der Benutzer kommt immer heraus. Speichern behält den Standardwert `true`. Ihre Lab-Notiz hält fest, welchen Modus Sie gewählt haben und warum Abbrechen trotzdem funktioniert.

`ValidateChildren()` als Schutz vor dem Speichern

Die Feldvalidierung läuft nur für Felder, die der Benutzer tatsächlich verlässt. Ein Benutzer kann einen Namen eingeben und auf Speichern klicken, ohne jemals `txtEmail` zu betreten, sodass dessen `Validating`-Handler nie gelaufen ist. Deshalb lautet die erste Anweisung von `btnSave_Click` `if (!this.ValidateChildren()) return;`. `ValidateChildren()` löst die Validierung für die untergeordneten Controls des Formulars aus, ob besucht oder nicht, zeigt jede Meldung an, die ihre Handler setzen, und gibt `false` zurück, wenn einer davon abgebrochen hat. Eine Überladung nimmt `ValidationConstraints` entgegen, um einzuschränken, welche untergeordneten Controls geprüft werden, zum Beispiel nur aktivierte Controls, auswählbare Controls, Tabstopps oder direkte Kind-Controls; das hilft, wenn ein Teil des Formulars ausgeblendet oder deaktiviert ist.

`Validate()` ist das kleinere Werkzeug: Es validiert das Control, das gerade den Fokus verliert, was nützlich ist, wenn `AutoValidate` auf `Disable` steht und Sie nur das aktuelle Feld prüfen möchten. Nachdem `ValidateChildren()` bestanden ist, setzen Sie das eigentliche Speichern in ein `try / catch`, damit ein fehlgeschlagener Schreibvorgang eine verständliche Systemmeldung anzeigt und nie mit einem korrigierbaren Eingabefehler verwechselt wird. Die Feldvalidierung schützt das Bearbeiten; die Validierung beim Speichern schützt das Speichern. Sie brauchen beides.



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:

- ✓ erklären, was Validierung in einer serverseitigen Wisej.NET-UI bedeutet.
- ✓ Validating und Validated an Editor-Controls behandeln.
- ✓ ungültige Eingaben abweisen, indem Sie `CancelEventArgs.Cancel` setzen.
- ✓ einen `AutoValidate`-Modus für ein Formular oder User-Control wählen.
- ✓ `Validate` und `ValidateChildren` zum richtigen Zeitpunkt aufrufen.

Lab- und Prüfungsleitfaden

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



Praxis-Lab

Legen Sie in ValidationClinic das Formular zur Kundenaufnahme mit den TextBoxen txtName und txtEmail, einem Speichern-Button und einem Abbrechen-Button an. Hängen Sie Validating-Handler an, die `e.Cancel = true` setzen und eine ErrorProvider-Meldung ausgeben, wenn der Name leer ist oder die E-Mail-Adresse kein @ enthält, die Meldung bei einem gültigen Wert wieder entfernen und den Namen in seinem Validated-Handler trimmen. Legen Sie den AutoValidate-Modus des Formulars fest, sorgen Sie dafür, dass Abbrechen keine Validierung auslöst, damit der Benutzer das Formular jederzeit verlassen kann, und rufen Sie `ValidateChildren()` am Anfang von `btnSave_Click` auf, damit das Speichern abbricht, sobald ein Feld ungültig ist. Schließen Sie mit einer kurzen Lab-Notiz ab, die das gewählte Fokusverhalten erklärt.

Ausgangspunkt

ValidationClinic ist die Anwendung, die Sie selbst erstellen, Modul für Modul — es gibt keinen Download, und es wird keine Startlösung für Sie installiert. Legen Sie in Visual Studio ein neues Wisej.NET-4-Projekt namens `ValidationClinic` an, geben Sie ihm eine Hauptseite, ein bearbeitbares Kontakt-Grid und ein einfaches In-Memory-Repository für Kontakte hinter diesem Grid, und validieren Sie noch nirgends etwas; jedes spätere Modul erweitert genau dieses Projekt. Zwei Abkürzungen, falls Sie das Grundgerüst nicht von Hand aufbauen möchten: Die **KI-Coding-Übung** dieses Moduls übergibt ChatGPT oder Claude einen fertigen Prompt, der die gesamte Lösung aus der Modulspezifikation erstellt, und unser eigener fertiger Build für jedes Modul dieses Kurses liegt im öffentlichen Beispiel-Repository `LearnWisej-Samples` — ein Ordner pro Modul; `Module 1` enthält genau dieses Grundgerüst und die Lösung zu diesem Lab. In diesem Lab fügen Sie das Formular zur Kundenaufnahme hinzu und geben ihm seine ersten beiden Validierungsebenen: die Feldvalidierung an jedem Editor und die Validierung beim Speichern am Speichern-Button.

Vorgeschlagenes Vorgehen

- ✓ Fügen Sie ein `CustomerIntakeForm` hinzu und platzieren Sie im Designer `txtName`, `txtEmail`, `btnSave`, `btnCancel` und ein Label `lblStatus`. Ziehen Sie einen `ErrorProvider` auf das Formular und setzen Sie seinen `ContainerControl` auf das Formular.
- ✓ Behandeln Sie `txtName.Validating` und `txtEmail.Validating`. Prüfen Sie jeweils den Wert; ist er falsch, setzen Sie `e.Cancel = true` und rufen `errorProvider.SetError` mit einer verständlichen Meldung auf, ist er richtig, rufen Sie `SetError` mit einem leeren String auf.
- ✓ Behandeln Sie `txtName.Validated` und trimmen Sie dort den Text, damit der Wert erst aufgeräumt wird, nachdem er akzeptiert wurde.
- ✓ Setzen Sie `AutoValidate` des Formulars und dann `btnCancel.CausesValidation = false`. Starten Sie das Formular, lassen Sie den Namen leer und klicken Sie auf Abbrechen, um zu zeigen, dass der Benutzer das Formular trotzdem verlassen kann.
- ✓ Machen Sie `if (!this.ValidateChildren()) return;` zur ersten Zeile von `btnSave_Click` und speichern Sie dann innerhalb von `try / catch`. Testen Sie Speichern mit einem Feld, in das

Sie nie geklickt haben, und schreiben Sie die Lab-Notiz.

Geforderte Ergebnisse

- ✓ Formular zur Kundenaufnahme mit txtName und txtEmail, einem Speichern-Button und einem Abbrechen-Button
- ✓ Validating-Handler, die für einen leeren Namen und eine ungültige E-Mail-Adresse `e.Cancel = true` und eine Fehlermeldung setzen
- ✓ Validated-Handler, der den akzeptierten Namen trimmt
- ✓ Speichern-Handler, der `ValidateChildren()` aufruft und abbricht, wenn es `false` zurückgibt
- ✓ Lab-Notiz, die den gewählten `AutoValidate`-Modus nennt und erklärt, warum Abbrechen auch bei einem ungültigen Formular funktioniert

Wiederholungsfragen

- ✓ Sie entfernen `e.Cancel = true` aus `txtEmail_Validating`, behalten aber den Aufruf von `setError`. Beschreiben Sie genau, was der Benutzer sieht, wenn er „`maria.example.com`“ mit Tab verlässt und dann auf Speichern klickt.
- ✓ Welche Ihrer Prüfungen laufen, während der Benutzer bearbeitet, und welche laufen nur, weil Speichern `ValidateChildren()` aufgerufen hat? Nennen Sie eine Benutzeraktion, die Speichern mit einem Feld erreicht, dessen Handler nie gelaufen ist.
- ✓ Sie stellen das Formular von `EnablePreventFocusChange` auf `EnableAllowFocusChange` um. Was ändert sich für den Benutzer, was bleibt gleich, und spielt `btnCancel.CausesValidation = false` dann noch eine Rolle?



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.