

Validation personnalisée : méthodes, services et classes ValidationRule

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Les règles métier que les règles intégrées ne savent pas exprimer : des méthodes de validation qui renvoient une chaîne, un service ContactValidator testable sans l'interface, une classe ValidationRule personnalisée avec OnValidating, OnValidated et OnControlCreated, et des règles inter-champs qui signalent chaque contrôle concerné. Ce que cela signifie pour un développeur qui connaît les bases de C# et débute dans le développement de formulaires Wisej.NET — et comment aborder ce module.



Objectif production

Les règles métier que les règles intégrées ne savent pas exprimer : des méthodes de validation qui renvoient une chaîne, un service ContactValidator testable sans l'interface, une classe ValidationRule personnalisée avec OnValidating, OnValidated et OnControlCreated, et des règles inter-champs qui signalent chaque contrôle concerné.

1. Le problème en production

Après le Module 3, le formulaire d'inscription de ValidationClinic sait qu'une adresse e-mail a besoin d'un @ et qu'un âge doit être un nombre entier. Il enregistre pourtant sans broncher un contact dont la date de début est postérieure à la date de fin, ou une date de naissance qui donne onze ans à la personne. Les règles intégrées vérifient la *forme* d'une valeur ; les vraies applications ont aussi besoin de règles sur le *sens*, et certaines de ces règles impliquent plusieurs contrôles. Ce module donne à ces règles une vraie place, pour qu'elles soient écrites une seule fois, affichées clairement et jamais ignorées sur Save.

2. Concepts clés

Concept	Pourquoi c'est important
méthode de validation personnalisée	Une méthode du formulaire qui répond à une seule question et renvoie une chaîne vide quand les données sont valides, ou un message en langage simple quand elles ne le sont pas.
service de validation	Une classe comme <code>ContactValidator</code> qui vérifie un modèle sans toucher à aucun contrôle, pour que les mêmes règles s'exécutent depuis le formulaire, une grille ou un test unitaire.
Record <code>ValidationMessage</code>	Un tout petit type de résultat contenant un nom de champ <code>Field</code> et un <code>Message</code> ; le service en renvoie une liste au lieu de définir les erreurs lui-même.
<code>ApplyModelValidation</code>	La méthode du formulaire qui efface les anciennes erreurs, associe chaque nom de champ à son contrôle avec <code>errorProvider.SetError</code> , remplit le récapitulatif et indique si tout est passé.
<code>ValidationRule</code>	La classe de base Wisej.NET derrière chaque règle de l'extender Validation ; en dériver transforme votre propre vérification en règle que vous pouvez attacher à n'importe quel contrôle.
<code>OnValidating</code> / <code>OnValidated</code> / <code>OnControlCreated</code>	Les trois redéfinitions d'une classe de règle : tester la valeur et renvoyer true ou false, agir après qu'une valeur est passée, et préparer le terrain une fois que le contrôle existe dans le navigateur.
règle inter-champs	Une règle dont la réponse dépend de deux contrôles ou plus, comme une date de début antérieure à la date de fin, si bien que son erreur doit apparaître sur chaque contrôle concerné.

3. Déroulé de la leçon

Commencer par une méthode qui renvoie un message

La règle personnalisée la plus simple est une méthode C# ordinaire. Elle lit les valeurs dont elle a besoin, pose une seule question et renvoie une chaîne : vide quand tout va bien, une phrase sur laquelle l'utilisateur peut agir dans le cas contraire. Dans le lab, `ValidateStartEndDates` lit `dtpStart.Value` et `dtpEnd.Value`, renvoie « Both dates are required. » quand l'une des deux manque, « Start date must be before end date. » quand l'ordre est incorrect, et "" sinon.

Remarquez ce que la méthode ne fait *pas* : elle ne touche pas à l'`ErrorProvider`. Une seconde petite méthode prend le texte renvoyé et le transmet à `errorProvider.SetError` pour chaque sélecteur. Comme un message vide efface une erreur, les deux mêmes lignes affichent le problème et le retirent une fois corrigé. Séparer la décision (la chaîne) de l'affichage (les appels à `SetError`) est le réflexe qui facilite l'étape suivante.

Déplacer les règles répétées dans un service de validation

Dès que la même vérification est nécessaire à un second endroit, par exemple le formulaire d'inscription et une boîte de dialogue d'ajout rapide, une méthode sur un seul formulaire ne suffit plus. Le lab déplace les règles sur le nom, l'e-mail et la date de naissance dans une classe `ContactValidator`. Sa méthode `Validate` reçoit un `ContactEditModel`, un simple objet avec les propriétés `Name`, `Email` et `BirthDate`, et renvoie une liste de records `ValidationMessage(Field, Message)` : « Name is required. », « Email must contain @. », « Birth date cannot be in the future. ».

Le service ne sait rien des zones de texte ; vous pouvez donc l'appeler depuis un test avec un modèle construit à la main et vérifier la liste qu'il renvoie, sans navigateur ni session Wisej.NET en cours. Le formulaire n'a alors besoin que d'un adaptateur, `ApplyModelValidation`. Il appelle `errorProvider.Clear()`, exécute le validateur, utilise un `switch` pour transformer `"Name"`, `"Email"` et `"BirthDate"` en `txtName`, `txtEmail` et `dtbBirthDate`, définit chaque erreur, regroupe tous les messages dans `validationSummaryLabel` et renvoie true uniquement si la liste est vide. Un message sans contrôle correspondant atteint tout de même le récapitulatif, donc rien n'est perdu. Le Module 5 construira le modèle par liaison de données ; ici, vous le remplissez vous-même à partir des contrôles.

Créer une classe dérivée de ValidationRule

Certaines règles appartiennent à un seul type de contrôle et doivent voyager d'un formulaire à l'autre comme les règles intégrées. Pour celles-ci, dérivez de `ValidationRule`. `MinimumAgeValidationRule` possède une propriété `MinimumAge` qui vaut 18 par défaut, et son constructeur appelle `base("MinimumAge", "The person is too young.")`, ce qui donne à la règle son nom et le message d'invalidité que l'extender affiche quand la règle échoue.

Vous l'attachez exactement comme les règles du Module 3, en passant une instance à `validation.SetValidationRules` pour `dtbBirthDate`. Dès lors, l'extender Validation l'exécute chaque fois que le sélecteur est validé, quand l'utilisateur quitte le champ et quand `ValidateChildren()` s'exécute sur Save, et achemine le message vers le même `ErrorProvider` que toutes les autres règles.

À quoi servent OnValidating, OnValidated et OnControlCreated

`OnValidating(Control control)` est le cœur de la règle. Elle renvoie `true` quand la valeur est valide et `false` dans le cas contraire ; elle ne renvoie pas de message, car celui-ci a été fourni au constructeur. La version du lab vérifie que le contrôle est un `DateTimePicker` dont la `Value` est un `DateTime`, calcule l'âge et retire un an si l'anniversaire n'est pas encore passé cette année :

```
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;    // nothing to check: Required decides about empty values
}
```

Les deux autres redéfinitions sont facultatives. `OnValidated` s'exécute après qu'une valeur est passée, ce qui en fait l'endroit pour le travail de suivi, comme la mise en forme de la valeur acceptée.

`OnControlCreated` s'exécute une fois que le contrôle a été créé côté client ; toute préparation qui nécessite l'existence du widget dans le navigateur doit donc s'y trouver plutôt que dans le constructeur. Pour une règle de débutant, redéfinir uniquement `OnValidating` est la norme.

Des règles inter-champs qui ne déroutent pas les utilisateurs

Une date de début postérieure à une date de fin n'est la faute d'aucun des deux sélecteurs pris isolément ; n'en accusez donc pas un seul. Définissez le même message sur `dtpStart` et `dtpEnd`, ajoutez une ligne au récapitulatif, et effacez les deux dès que l'ordre est correct. Évitez `e.Cancel = true` pour ce type de règle : cela bloquerait le focus dans le premier sélecteur alors que la correction se trouve peut-être dans le second.

Faites ensuite de Save le dernier point de contrôle. `btnSave_Click` exécute d'abord les règles de champ avec `ValidateChildren()`, ce qui inclut `MinimumAgeValidationRule`. Si cela passe, il construit le modèle et appelle `ApplyModelValidation`, puis exécute la vérification des dates, car sinon `errorProvider.Clear()` effacerait les erreurs de dates. Ce n'est que lorsque toutes les couches passent qu'il appelle le repository. Une règle en échec est un retour produit normal, pas une exception ; la base de données n'est donc jamais ouverte pour des données dont vous savez déjà qu'elles sont fausses.



Réflexe des couches de validation

Nommez la couche avant d'écrire la vérification. La validation des champs protège la saisie et s'exécute quand l'utilisateur quitte un contrôle ; la validation du modèle protège les données et applique les mêmes règles depuis le formulaire, la grille et n'importe quel service ; la validation à l'enregistrement protège la base de données et réexécute toutes les couches avant que quoi que ce soit ne soit écrit. Affichez chaque erreur près de son champ et dans un récapitulatif visible, dans un langage simple qui dit quoi faire ensuite, et séparez les erreurs que l'utilisateur peut corriger des pannes système.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Écrire des méthodes de validation personnalisées pour des règles métier de niveau débutant.

- ✓ Déplacer les règles répétées dans un service de validation réutilisable.
- ✓ Créer une classe personnalisée dérivée de `ValidationRule`.
- ✓ Comprendre `OnValidating`, `OnValidated` et `OnControlCreated`.
- ✓ Appliquer une validation inter-champs sans dérouter les utilisateurs.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.

Lab pratique

Ajoutez une validation personnalisée à ValidationClinic en trois étapes. Écrivez `ValidateStartEndDates` comme une méthode qui renvoie une chaîne vide quand les données sont valides et un message quand la date de début est postérieure à la date de fin, et définissez ce message à la fois sur `dtoStart` et sur `dtoEnd`. Déplacez les règles sur le nom, l'e-mail et la date de naissance dans un service `ContactValidator` qui renvoie une liste de records `ValidationMessage`, et associez chaque nom de champ à son contrôle dans `ApplyModelValidation`. Créez ensuite `MinimumAgeValidationRule`, dérivée de `ValidationRule`, avec une propriété `MinimumAge`, redéfinissez `OnValidating` pour calculer l'âge à partir du `DateTimePicker`, attachez-la à `dtoBirthDate` via l'extender `Validation`, et exécutez le service dans `btnSave_Click` avant tout enregistrement.

Point de départ

Ouvrez ValidationClinic tel que vous l'avez laissé après le Module 3 : le formulaire d'inscription possède un extender `Validation` et un `ErrorProvider` qui vérifient l'e-mail, l'âge, le téléphone, la limite de crédit et le code client, ainsi qu'un `validationSummaryLabel`. Les sélecteurs `dtoBirthDate`, `dtoStart` et `dtoEnd` n'ont encore aucune règle, et `btnSave_Click` enregistre tout ce que `ValidateChildren()` laisse passer.

Approche suggérée

- ✓ Écrivez `ValidateStartEndDates` pour qu'elle renvoie "" ou un message, puis un petit utilitaire qui transmet cette même chaîne à `errorProvider.SetError` pour `dtoStart` et pour `dtoEnd`. Essayez un ordre incorrect, puis corrigez-le, et vérifiez que les deux icônes apparaissent puis disparaissent toutes les deux.
- ✓ Créez un record `ValidationMessage(Field, Message)` et une classe `ContactValidator` dont la méthode `Validate` vérifie dans un `ContactEditModel` un nom vide, une adresse e-mail absente ou sans @ et une date de naissance dans le futur. Elle ne doit référencer aucun contrôle.
- ✓ Ajoutez `ApplyModelValidation` au formulaire : effacez l' `ErrorProvider`, exécutez le validateur, associez `"Name"`, `"Email"` et `"BirthDate"` à leurs contrôles, remplissez le libellé récapitulatif et renvoyez si la liste était vide.
- ✓ Créez `MinimumAgeValidationRule`, dérivée de `ValidationRule`, appelez `base("MinimumAge", "The person is too young.")`, redéfinissez `OnValidating` pour calculer l'âge, et attachez une instance à `dtoBirthDate` avec `validation.SetValidationRules`.
- ✓ Réécrivez `btnSave_Click` par couches : arrêtez-vous si `ValidateChildren()` échoue, arrêtez-vous si `ApplyModelValidation` ou la vérification des dates échoue, et seulement ensuite appelez le repository dans un `try/catch` pour les vraies pannes système. Ajoutez à votre fichier de lab une courte note qui nomme chaque API utilisée.

Livrables attendus

- ✓ Méthode `ValidateStartEndDates` qui définit la même erreur sur les deux sélecteurs de date et les efface tous les deux une fois corrigés
- ✓ Service `ContactValidator` renvoyant des records `ValidationMessage` pour le nom, l'e-mail et la date de naissance
- ✓ Méthode `ApplyModelValidation` associant chaque nom de champ à son contrôle et au récapitulatif
- ✓ Classe `MinimumAgeValidationRule` redéfinissant `OnValidating`, attachée à `dtpBirthDate`
- ✓ Gestionnaire de `Save` qui exécute les règles de champ, puis le service de validation, et seulement ensuite enregistre

Questions de révision

- ✓ La vérification de l'âge minimal pourrait se trouver dans `ContactValidator` ou dans `MinimumAgeValidationRule`. Qu'est-ce qui change pour l'utilisateur dans chaque cas, et quand voudriez-vous les deux ?
- ✓ Votre gestionnaire de `Save` appelle `ApplyModelValidation` après avoir défini les erreurs de dates, et les icônes des dates n'apparaissent jamais. Expliquez pourquoi, et quel ordre corrige le problème.
- ✓ Comment prouveriez-vous que `ContactValidator` rejette une adresse e-mail sans @ sans démarrer l'application ni ouvrir de navigateur ?



Réflexe des couches de validation

Nommez la couche avant d'écrire la vérification. La validation des champs protège la saisie et s'exécute quand l'utilisateur quitte un contrôle ; la validation du modèle protège les données et applique les mêmes règles depuis le formulaire, la grille et n'importe quel service ; la validation à l'enregistrement protège la base de données et réexécute toutes les couches avant que quoi que ce soit ne soit écrit. Affichez chaque erreur près de son champ et dans un récapitulatif visible, dans un langage simple qui dit quoi faire ensuite, et séparez les erreurs que l'utilisateur peut corriger des pannes système.