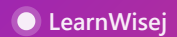


Exercice de code avec l'IA

Construisez l'exemple ValidationClinic de ce module avec ChatGPT ou Claude à partir d'un prompt prêt à l'emploi qui oriente le modèle vers la spécification du module, puis relisez, exécutez et étendez ce qu'il vous renvoie — y compris en le comparant à notre propre solution de référence sur GitHub.



Écrire le code, c'est désormais la moitié facile.

Obtenir d'un assistant moderne du code qui fonctionne pour « Validation personnalisée : méthodes, services et classes ValidationRule » ne demande qu'un bon prompt. Le juger, le corriger et l'étendre demande tout ce que ce module a couvert. C'est sur ce partage que repose l'exercice : vous apportez la spécification et la relecture, l'assistant apporte la frappe.

Ce que vous demandez

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 dtpStart et sur dtpEnd. 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 dtpBirthDate via l'extender Validation, et exécutez le service dans btnSave_Click avant tout enregistrement.

Le module demande ces livrables, et le prompt ci-dessous demande exactement ceux-là à l'assistant :

- ✓ 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

Pourquoi il est conçu ainsi

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.

Voici les termes que l'assistant doit maîtriser. Si ce qu'il vous renvoie les emploie de façon approximative, il n'a pas compris le module :

Terme	Ce qu'il signifie ici
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 ValidationMessage	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.
ApplyModelValidation	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.
ValidationRule	La classe de base <code>Wisej.NET</code> derrière chaque règle de l'extender <code>Validation</code> ; en dériver transforme votre propre vérification en règle que vous pouvez attacher à n'importe quel contrôle.
OnValidating / OnValidated / OnControlCreated	Les trois redéfinitions d'une classe de règle: tester la valeur et renvoyer <code>true</code> ou <code>false</code> , 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.

Construisez-le — choisissez votre assistant

Les deux boutons ouvrent un nouvel onglet avec le **même prompt prêt à l'emploi** déjà rempli. Le prompt oriente le modèle vers la spécification de ce module — un unique fichier Markdown qui contient la leçon, l'énoncé du lab et les livrables — pour qu'il travaille sur le vrai contenu au lieu de deviner de quoi parle le Module 4. Vous pouvez ouvrir vous-même [module-04.md](#) pour voir exactement ce qu'il lira.



Utiliser ChatGPT

Ouvre chatgpt.com avec le prompt de ce module déjà chargé. Fonctionne avec GPT-5 ou Codex ; toute offre capable de naviguer sur le web récupérera la spécification.



Utiliser Claude

Ouvre claude.ai avec le même prompt déjà chargé. Si vous utilisez Claude Code, vous pouvez plutôt le coller dans un terminal ouvert sur un dossier vide.



Envoyez-le tel quel et vous obtenez un projet, pas un extrait

Le prompt se termine en demandant toute la solution sous forme d'un unique `.zip`, avec les étapes pour la compiler et l'exécuter. Téléchargez-le, ouvrez la solution dans Visual Studio, appuyez sur F5 et ValidationClinic s'ouvre sur `localhost`. Il vous faut le SDK .NET et le package NuGet `Wisej-4` ; tout le reste, le prompt demande à l'assistant de l'écrire lui-même.

Je suis le Module 4 – « Validation personnalisée : méthodes, services et classes ValidationRule » – du cours LearnWisej « Validation dans Wisej.NET », et je veux construire avec toi l'exemple de ce module. Réponds-moi en français.

Commence par lire la spécification du module. C'est un unique fichier Markdown (en anglais) qui contient la leçon complète, l'énoncé du lab et les livrables :

<https://learnwisej.com/assets/courses/validation-in-wisej-net/specs/module-04.md>

Construis ensuite un projet Wisej.NET 4 complet que je puisse ouvrir dans Visual Studio :

1. Projet « ValidationClinic », C#, net8.0 ou ultérieur, package NuGet Wisej-4 4.1.x, hôte Kestrel qui appelle app.UseWisej().
2. Construis ce que demande le lab de la spécification : 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 dtpStart et sur dtpEnd. 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 dtpBirthDate via l'extender Validation, et exécutez le service dans btnSave_Click avant tout enregistrement.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - 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
4. Suis la spécification partout où elle nomme une classe, une propriété ou un événement Wisej.NET – utilise exactement celui-là. Là où elle laisse le choix, dis-moi quelle API tu as choisie et pourquoi.
5. Garde l'autorité côté serveur : l'état et la validation vivent dans .NET, le navigateur affiche et remonte les informations. Ne déplace jamais une décision dans JavaScript parce que c'est plus simple là-bas.
6. Rends le résultat démontrable : je dois pouvoir l'exécuter et voir le comportement dont traite ce module, y compris ce qui se passe quand quelque chose tourne mal.

Avant d'écrire la moindre ligne de code, reformule avec tes propres mots ce que tu vas construire et où vivra chaque élément, pour que je puisse corriger la conception tant que cela ne coûte pas cher.

Quand le code est terminé, rassemble toute la solution dans un unique fichier .zip téléchargeable et donne-moi les étapes exactes pour la compiler et l'exécuter.

Enfin, réponds à ces questions de la spécification en t'appuyant sur ta propre implémentation :

- 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?

Pose-moi des questions sur tout ce qui est ambigu avant de commencer.

Puis relisez ce qui vous revient

Un code qui compile n'est pas encore un code juste. Les questions intéressantes sont des questions de conception : est-ce bien le contrôle Wisej.NET que la tâche exige, suit-il le cycle de vie que le framework vous offre, l'état de référence est-il sur le serveur ou a-t-il été laissé dans le navigateur parce que c'était pratique ? Lisez le résultat en relecteur, pas en client.

Ce module vous donne la checklist. Répondez à chaque point en vous appuyant sur le code que vous venez de recevoir :

- ✓ 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?

Tout ce que vous avez dû corriger mérite d'être noté. Une courte note des conventions et des décisions retenues, collée dans le premier message de votre prochaine session, vaut plus que n'importe quel prompt astucieux.



Nous l'avons déjà construit — comparez votre version à la nôtre

L'implémentation de référence du Module 4 se trouve dans le dépôt public des exemples. Construisez d'abord la vôtre, puis ouvrez la nôtre : ce sont les différences qui sont intéressantes.



[Ouvrir l'exemple du Module 4 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Validation in Wisej.NET Course/Module 4`.



Notre exemple est un point de départ, pas un produit fini

Il a été construit de la même manière que vous construisez le vôtre — par prompt, puis corrigé — et il s'arrête volontairement là où s'arrête le module. Attendez-vous à des éléments provisoires là où irait du code de production, à une gestion des erreurs minimale et à l'absence de tests. Voyez-y une invitation : utilisez ce que vous avez appris dans ce module, avec l'assistant, pour aller plus loin. Améliorer du code que vous n'avez pas écrit, c'est précisément la compétence que ce cours enseigne.

Trois prompts pour aller plus loin

Orientez votre assistant vers le projet que vous venez de construire — ou vers le nôtre et essayez ceux-ci. Chacun est un exercice de relecture et d'extension, pas une réécriture.

- ✓ **Prouvez que ça fonctionne vraiment.** *"Passe en revue ce projet livrable par livrable par rapport à la spécification du module sur <https://learnwisej.com/assets/courses/validation-in-wisej-net/specs/module-04.md> et montre-moi la preuve que chacun est réellement terminé — le chemin dans le code, où cliquer et ce que je dois voir. Là où un livrable n'est que partiellement satisfait, dis-le clairement au lieu de le défendre. Ensuite, ajoute ce qui manque."*
- ✓ **Relisez-le comme une pull request.** *"Fais-moi parcourir ce code ValidationClinic comme si tu relisais la pull request d'un collègue et que tu ne l'avais pas écrit. Où vit l'état, que se passe-t-il de bout en bout lors d'une action de l'utilisateur, que renverrais-tu pour correction, et quelles parties ne serais-tu pas à l'aise de montrer à un client ? Sois précis et cite les lignes dont tu parles — ne me résume pas la conception."*
- ✓ **Remplacez une pièce sans déplacer les jointures.** *"Choisis la partie de ce projet ValidationClinic qui remplace le plus manifestement quelque chose de réel — une source de données factice, un stockage en mémoire, un service simplifié — et remplace-la par une implémentation digne de la production. Le contrat de part et d'autre ne doit pas changer : mêmes types, mêmes appels, même comportement en cas d'échec. Dis-moi ce que l'implémentation réelle t'a obligé à gérer et que la version provisoire ne gérait pas."*



Emportez-le dans le lab

Le lab pratique qui clôt ce module demande les mêmes livrables que ceux que vous venez de faire produire à l'assistant. Servez-vous de ce que vous avez ici comme forme de votre réponse et comme preuve à l'appui — pas comme la réponse elle-même. Le lab est noté sur votre raisonnement, et cette partie doit venir de vous.