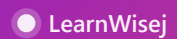


Exercice de code avec l'IA

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.



Le même module, construit comme vous construirez vraiment.

Toutes les autres sections de ce module expliquent comment fonctionne « Validation liée aux données et erreurs du modèle ». Celle-ci vous demande de le produire — en pilotant un assistant avec la spécification du module elle-même, puis en vérifiant que le résultat s'exécute et fait ce que décrit l'énoncé du lab. C'est la méthode de travail que vous utiliserez sur de vrais projets, mise en pratique sur un sujet dont vous connaissez déjà la bonne réponse.

Ce que vous demandez

Liez le formulaire d'inscription de ValidationClinic à un ContactEditModel via un contactBindingSource et ajoutez les annotations Required, StringLength, EmailAddress et Range à ses propriétés Name, Email et Age. Dans btnSave_Click, appelez d'abord contactBindingSource.EndEdit(), exécutez Validator.TryValidateObject avec validateAllProperties à true, et associez chaque nom de membre des ValidationResult à son contrôle via une table de correspondance des champs et le récapitulatif de validation. Implémentez ensuite IDataErrorInfo sur un second modèle, faites pointer l'ErrorProvider vers la binding source avec BindToDataAndErrors, et montrez les mêmes erreurs apparaître par liaison de données sans appel à SetError écrit à la main.

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

- ✓ ContactEditModel avec DataAnnotations, lié au formulaire via un BindingSource
- ✓ Gestionnaire de Save qui appelle EndEdit avant d'exécuter Validator.TryValidateObject
- ✓ Table de correspondance des champs qui transforme les noms de membres des ValidationResult en erreurs de contrôle et en lignes du récapitulatif
- ✓ Modèle IDataErrorInfo dont les erreurs apparaissent via un ErrorProvider lié aux données
- ✓ Appel à BindToDataAndErrors utilisé quand la source de données du provider change à l'exécution

Pourquoi il est conçu ainsi

Tout ce que ValidationClinic vérifie jusqu'ici vit dans le formulaire d'inscription: un gestionnaire Validating ici, une règle sur l'extender Validation là, un appel à ContactValidator dans btnSave_Click. Mais un contact n'arrive pas uniquement par ce formulaire. Il peut être modifié dans la grille des contacts, importé depuis

un fichier ou créé par un service, et aucun de ces chemins n'exécute les gestionnaires du formulaire. Ce module déplace les règles de base sur le `ContactEditModel` lui-même, pour que les données portent leur propre définition de « valide » et que chaque écran puisse leur poser la même question.

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
<code>BindingSource</code>	Un composant qui se place entre les contrôles et votre objet: son <code>DataSource</code> est le modèle, et chaque contrôle se lie à une propriété par son intermédiaire au lieu d'être recopié à la main.
<code>EndEdit()</code>	Pousse la valeur que l'utilisateur est encore en train de modifier du contrôle vers le modèle, pour que la validation lise ce qui est à l'écran plutôt que la dernière valeur validée.
<code>DataAnnotations</code>	Des attributs comme <code>[Required]</code> , <code>[StringLength]</code> , <code>[EmailAddress]</code> et <code>[Range]</code> placés sur les propriétés du modèle, chacun avec un <code>ErrorMessage</code> rédigé pour l'utilisateur.
<code>Validator.TryValidateObject</code>	La méthode.NET qui applique ces attributs à un objet et remplit une liste d'objets <code>ValidationResult</code> ; elle ne sait rien des contrôles ni de Wisej.NET.
<code>MemberNames</code>	Les noms des propriétés auxquelles appartient un <code>ValidationResult</code> , seul lien entre un message d'erreur et le contrôle qui doit l'afficher.
<code>IDataErrorInfo</code>	Une interface qui permet à un modèle de répondre à « qu'est-ce qui ne va pas avec cette propriété? » au moyen d'un indexeur, pour que la couche de liaison demande les erreurs au lieu que votre code les pousse.
ErrorProvider lié aux données	Un <code>ErrorProvider</code> dont le <code>DataSource</code> est la même binding source que celle des contrôles, si bien qu'il affiche les erreurs du modèle à côté des contrôles liés sans aucun appel à <code>SetError</code> .

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 5. Vous pouvez ouvrir vous-même [module-05.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



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.

navigation sur le web récupérera la spécification.



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 5 - « Validation liée aux données et erreurs du modèle » - 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-05.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 : Liez le formulaire d'inscription de ValidationClinic à un `ContactEditModel` via un `contactBindingSource` et ajoutez les annotations `Required`, `StringLength`, `EmailAddress` et `Range` à ses propriétés `Name`, `Email` et `Age`. Dans `btnSave_Click`, appelez d'abord `contactBindingSource.EndEdit()`, exécutez `Validator.TryValidateObject` avec `validateAllProperties` à `true`, et associez chaque nom de membre des `ValidationResult` à son contrôle via une table de correspondance des champs et le récapitulatif de validation. Implémentez ensuite `IDataErrorInfo` sur un second modèle, faites pointer l'`ErrorProvider` vers la binding source avec `BindToDataAndErrors`, et montrez les mêmes erreurs apparaître par liaison de données sans appel à `SetError` écrit à la main.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - `ContactEditModel` avec `DataAnnotations`, lié au formulaire via un `BindingSource`
 - Gestionnaire de `Save` qui appelle `EndEdit` avant d'exécuter `Validator.TryValidateObject`
 - Table de correspondance des champs qui transforme les noms de membres des `ValidationResult` en erreurs de contrôle et en lignes du récapitulatif
 - Modèle `IDataErrorInfo` dont les erreurs apparaissent via un `ErrorProvider` lié aux données
 - Appel à `BindToDataAndErrors` utilisé quand la source de données du provider change à l'exécution
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 :

- Un testeur tape une nouvelle adresse e-mail et appuie sur Entrée pour enregistrer, et c'est l'ancienne adresse, valide, qui est vérifiée et stockée. Quelle ligne manque, et pourquoi le formulaire paraît-il correct à l'écran alors que le modèle est faux?
- Vous retirez `validateAllProperties: true` et chaque enregistrement « fonctionne » encore.

Laquelle de vos quatre annotations a cessé de s'exécuter, et quelles données incorrectes peuvent désormais atteindre le repository?

– Le service d'import crée des contacts sans ouvrir le formulaire. Quelles règles de ce module protègent encore ce chemin, et lesquelles ne s'exécutent que parce qu'un utilisateur a cliqué sur Save?

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 :

- ✓ Un testeur tape une nouvelle adresse e-mail et appuie sur Entrée pour enregistrer, et c'est l'ancienne adresse, valide, qui est vérifiée et stockée. Quelle ligne manque, et pourquoi le formulaire paraît-il correct à l'écran alors que le modèle est faux?
- ✓ Vous retirez `validateAllProperties: true` et chaque enregistrement « fonctionne » encore. Laquelle de vos quatre annotations a cessé de s'exécuter, et quelles données incorrectes peuvent désormais atteindre le repository?
- ✓ Le service d'import crée des contacts sans ouvrir le formulaire. Quelles règles de ce module protègent encore ce chemin, et lesquelles ne s'exécutent que parce qu'un utilisateur a cliqué sur Save?

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 5 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 5 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Validation in Wisej.NET Course/Module 5`.



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.

- ✓ **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 gèrait pas."*
- ✓ **Faites-le résister aux mauvais jours.** *"Examine ce projet ValidationClinic pour repérer les modes de défaillance qu'il ne gère pas encore — une saisie erronée ou manquante, une dépendance lente ou indisponible, un traitement qui aboutit après que l'utilisateur a changé de page, et deux actions qui se chevauchent. Pour chacun, dis-moi d'abord s'il s'agit d'un risque réel dans Wisej.NET 4 et pourquoi, et seulement ensuite écris la correction. Là où tu estimes qu'aucune gestion n'est nécessaire, dis-le et explique ton raisonnement au lieu d'ajouter du code défensif."*



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.