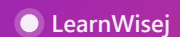


Exercice de code avec l'IA

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.



L'assistant, ce sont les mains rapides. L'architecte, c'est toujours vous.

Un assistant peut produire l'exemple de ce module en quelques minutes. Ce qu'il ne peut pas faire, c'est décider si le résultat est la bonne conception pour votre application — cette décision vous revient. C'est vous qui apportez le jugement : briefez-le correctement, puis vérifiez ce qu'il vous renvoie à la lumière de ce que le Module 7 vient de vous apprendre.

Ce que vous demandez

Terminez le projet de synthèse ValidationClinic. Écrivez btnSave_Click comme un pipeline fixe : réinitialisez le récapitulatif, appelez bindingSource.EndEdit(), arrêtez-vous si ValidateChildren() échoue, arrêtez-vous si la vérification DataAnnotations échoue, arrêtez-vous si ContactValidator renvoie des erreurs métier, arrêtez-vous si la grille des contacts porte encore des erreurs de ligne, et seulement ensuite enregistrez dans un try/catch qui transforme un nom en double en erreur de champ et toute autre exception en message compréhensible. Créez SummaryErrorProvider en implémentant IErrorProvider, affectez-le à validation.ErrorProvider, et faites passer les erreurs par un utilitaire SetError qui écrit à la fois dans l'ErrorProvider à icônes et dans le provider récapitulatif. Testez des valeurs obligatoires, invalides, valides et aux limites, et rédigez un court README qui explique la conception de la validation.

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

- ✓ Pipeline de Save qui exécute EndEdit, ValidateChildren, puis les vérifications du modèle, des règles métier et de la grille avant d'enregistrer
- ✓ Classe SummaryErrorProvider implémentant SetError et GetError d'IErrorProvider
- ✓ Utilitaire SetError qui écrit à la fois dans l'ErrorProvider à icônes et dans le provider récapitulatif
- ✓ try/catch autour de l'enregistrement qui distingue une erreur de nom en double des pannes système
- ✓ README avec la conception de la validation et des cas de test pour des valeurs obligatoires, invalides, valides et aux limites

Pourquoi il est conçu ainsi

Un utilisateur qui remplit le formulaire d'inscription de ValidationClinic ne se demande jamais quelle API a trouvé une erreur; il remarque seulement si l'écran lui dit ce qui ne va pas et le laisse terminer. Après six

modules, l'application dispose de gestionnaires de champ, de règles d'étendre, d'un service de validation, de modèles annotés et de règles de grille, mais chacun a été ajouté séparément, et rien ne garantit encore qu'ils s'exécutent tous, dans le bon ordre, à chaque clic sur Save. Ce module réunit les couches dans un pipeline de Save prévisible et un récapitulatif visible, puis vous demande de prouver la conception dans un projet de synthèse.

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
Pipeline de Save	Une séquence fixe dans btnSave_Click qui valide les modifications en attente, exécute tour à tour chaque couche de validation et sort à la première couche qui échoue, pour que rien n'atteigne le repository sans vérification.
Sortie anticipée	Chaque étape termine le gestionnaire avec return quand elle trouve des erreurs, pour que les étapes suivantes ne s'exécutent jamais sur des données qu'une étape précédente a déjà rejetées.
IErrorProvider	La petite interface à laquelle le système de validation de Wisej.NET envoie ses messages: SetError(Control, string) pour définir ou effacer un message et GetError(Control) pour le relire.
SummaryErrorProvider	Votre propre IErrorProvider, qui tient un dictionnaire de paires contrôle-message et réécrit à partir de celui-ci un libellé récapitulatif chaque fois qu'une erreur est définie ou effacée.
Double provider	Un unique utilitaire SetError qui écrit le même message dans l'ErrorProvider intégré (l'icône à côté du champ) et dans le provider récapitulatif (la liste en haut).
Erreur utilisateur ou panne système	Une erreur que l'utilisateur peut corriger, comme un nom en double, devient un message de champ; une exception inattendue est journalisée pour les développeurs et affichée sous forme de message générique et posé.
Test aux limites	Une valeur de test située exactement à la limite d'une règle, comme un âge de 0 ou de 120, qui détecte les erreurs de décalage d'une unité que les valeurs valides et invalides ordinaires laissent passer.

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 7. Vous pouvez ouvrir vous-même [module-07.md](#) pour voir exactement ce qu'il lira.



Utiliser ChatGPT



Utiliser Claude

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.

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 7 – « UX de validation en production, error providers personnalisés et projet de synthèse » – 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-07.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 : Terminez le projet de synthèse ValidationClinic. Écrivez `btnSave_Click` comme un pipeline fixe : réinitialisez le récapitulatif, appelez `bindingSource.EndEdit()`, arrêtez-vous si `ValidateChildren()` échoue, arrêtez-vous si la vérification `DataAnnotations` échoue, arrêtez-vous si `ContactValidator` renvoie des erreurs métier, arrêtez-vous si la grille des contacts porte encore des erreurs de ligne, et seulement ensuite enregistrez dans un `try/catch` qui transforme un nom en double en erreur de champ et toute autre exception en message compréhensible. Créez `SummaryErrorProvider` en implémentant `IErrorProvider`, affectez-le à `validation.ErrorProvider`, et faites passer les erreurs par un utilitaire `SetError` qui écrit à la fois dans l'`ErrorProvider` à icônes et dans le provider récapitulatif. Testez des valeurs obligatoires, invalides, valides et aux limites, et rédigez un court README qui explique la conception de la validation.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - Pipeline de Save qui exécute `EndEdit`, `ValidateChildren`, puis les vérifications du modèle, des règles métier et de la grille avant d'enregistrer
 - Classe `SummaryErrorProvider` implémentant `SetError` et `GetError` d'`IErrorProvider`
 - Utilitaire `SetError` qui écrit à la fois dans l'`ErrorProvider` à icônes et dans le provider récapitulatif
 - `try/catch` autour de l'enregistrement qui distingue une erreur de nom en double des pannes système
 - README avec la conception de la validation et des cas de test pour des valeurs obligatoires, invalides, valides et aux limites
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 dans le formulaire et clique immédiatement sur Save; c'est l'ancienne adresse qui est enregistrée. Quelle étape du pipeline manque ou est mal placée, et pourquoi sa position compte-t-elle?

– Votre SummaryErrorProvider affiche deux fois « Email is required. » après que l'utilisateur a corrigé puis à nouveau cassé le champ. Quelle partie de SetLastError devrait l'empêcher, et de quoi dépend-elle?

– Le repository lève une exception contenant un message de base de données quand le réseau tombe. Décrivez ce que voit l'utilisateur et ce que le développeur peut retrouver ensuite, et expliquez en quoi cela diffère du cas du nom en double.

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 dans le formulaire et clique immédiatement sur Save; c'est l'ancienne adresse qui est enregistrée. Quelle étape du pipeline manque ou est mal placée, et pourquoi sa position compte-t-elle?
- ✓ Votre SummaryErrorProvider affiche deux fois « Email is required. » après que l'utilisateur a corrigé puis à nouveau cassé le champ. Quelle partie de SetLastError devrait l'empêcher, et de quoi dépend-elle?
- ✓ Le repository lève une exception contenant un message de base de données quand le réseau tombe. Décrivez ce que voit l'utilisateur et ce que le développeur peut retrouver ensuite, et expliquez en quoi cela diffère du cas du nom en double.

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 7 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 7 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Validation in Wisej.NET Course/Module 7`.



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.

- ✓ **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."*
- ✓ **Ajoutez le deuxième cas.** *"Étends ce projet ValidationClinic avec un deuxième cas qui suit le même modèle que le premier — une autre entité, un autre écran ou un autre contrôle de même forme, câblé de la même manière. Extrais dans une classe de base ou un service ce qui est réellement partagé, mais seulement là où cela se justifie ; deux choses similaires n'appellent pas toujours une abstraction. Montre-moi le diff et justifie chaque élément que tu as extrait."*
- ✓ **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-07.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."*



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.