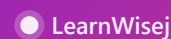


Exercice de code avec l'IA

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 2 vient de vous apprendre.

Ce que vous demandez

Transformez le formulaire de contact de ValidationClinic en formulaire à validation programmatique. Créez un seul ErrorProvider pour le formulaire avec ContainerControl défini, NeverBlink et une icône alignée à droite, écrivez un record FieldValidation ainsi que des utilitaires ValidateName, ValidateEmail et ValidatePhone qui renvoient un message au lieu de toucher à l'interface, et une méthode Apply qui appelle SetError pour chaque résultat et remplit un validationSummaryLabel visible uniquement tant qu'il existe des erreurs. Utilisez InvalidMessage sur le champ du code client pour comparer les deux styles de retour, et assurez-vous que corriger un champ n'efface que l'erreur de ce champ.

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

- ✓ ErrorProvider au niveau du formulaire avec ContainerControl, BlinkStyle et alignement des icônes configurés
- ✓ Record FieldValidation et méthodes utilitaires ValidateName, ValidateEmail et ValidatePhone
- ✓ Méthode Apply qui définit ou efface l'erreur de chaque contrôle et renvoie si tous les champs sont valides
- ✓ validationSummaryLabel listant tous les problèmes en cours et masqué quand il n'y en a aucun
- ✓ Un champ utilisant InvalidMessage au lieu d'ErrorProvider, avec une note comparant les deux

Pourquoi il est conçu ainsi

Le Module 1 a doté le formulaire d'inscription de ValidationClinic de deux gestionnaires Validating, et cela fonctionne pour deux champs. Un vrai formulaire de contact comporte un nom, une adresse e-mail, un numéro de téléphone, un code client et plus encore; si chaque champ reçoit son propre gestionnaire écrit à la main, avec sa propre manière d'afficher un message, le formulaire devient vite incohérent: un champ affiche une icône, un autre rien du tout, un troisième laisse une ancienne erreur à l'écran alors que

l'utilisateur l'a corrigée. Ce module remplace cette dérive par un seul modèle: de petites méthodes utilitaires décident de ce qui ne va pas, une méthode affiche le résultat, et un récapitulatif visible indique à chaque utilisateur ce qu'il doit corriger sans l'obliger à survoler quoi que ce soit.

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
ErrorProvider	Un composant extender qui dessine une icône d'erreur à côté de tout contrôle auquel vous attribuez un message et affiche ce message en infobulle; une seule instance sert tout le formulaire.
ContainerControl	Le formulaire ou le contrôle utilisateur auquel appartient le provider; le définir sur this indique au provider sur quelle surface dessiner ses icônes.
setError(control, message)	Affiche message à côté de control; l'appeler avec une chaîne vide ou null retire l'erreur de ce seul contrôle.
getError, hasErrors, clear()	Lire le message actuel d'un contrôle, demander s'il reste une erreur sur le formulaire, ou retirer toutes les erreurs d'un coup quand le formulaire entier est réinitialisé.
InvalidMessage	Une propriété des contrôles d'édition qui prennent en charge l'interface IValidation; le contrôle affiche lui-même le message dans son infobulle d'invalidité, sans provider distinct.
Record FieldValidation	Un tout petit type de résultat contenant le contrôle, un nom de champ et un message, où un message vide signifie valide; il permet aux règles de renvoyer des réponses au lieu de modifier l'écran.
validationSummaryLabel	Un simple Label qui liste en texte tous les problèmes en cours, visible uniquement tant qu'il existe des erreurs, pour que le retour ne dépende jamais du fait de voir ou de survoler une icône.

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 2. Vous pouvez ouvrir vous-même [module-02.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 2 - « Validation programmatique et retour avec ErrorProvider » - 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-02.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 : Transformez le formulaire de contact de ValidationClinic en formulaire à validation programmatique. Créez un seul `ErrorProvider` pour le formulaire avec `ContainerControl` défini, `NeverBlink` et une icône alignée à droite, écrivez un record `FieldValidation` ainsi que des utilitaires `ValidateName`, `ValidateEmail` et `ValidatePhone` qui renvoient un message au lieu de toucher à l'interface, et une méthode `Apply` qui appelle `SetError` pour chaque résultat et remplit un `validationSummaryLabel` visible uniquement tant qu'il existe des erreurs. Utilisez `InvalidMessage` sur le champ du code client pour comparer les deux styles de retour, et assurez-vous que corriger un champ n'efface que l'erreur de ce champ.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - `ErrorProvider` au niveau du formulaire avec `ContainerControl`, `BlinkStyle` et alignement des icônes configurés
 - Record `FieldValidation` et méthodes utilitaires `ValidateName`, `ValidateEmail` et `ValidatePhone`
 - Méthode `Apply` qui définit ou efface l'erreur de chaque contrôle et renvoie si tous les champs sont valides
 - `validationSummaryLabel` listant tous les problèmes en cours et masqué quand il n'y en a aucun
 - Un champ utilisant `InvalidMessage` au lieu d'`ErrorProvider`, avec une note comparant les deux
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 :

- Le gestionnaire `txtName_Validating` d'un collègue appelle `errorProvider.Clear()` dès que le nom devient valide. Qu'est-ce qui ne va pas pour un utilisateur qui a aussi saisi une

adresse e-mail invalide, et qu'est-ce que le gestionnaire devrait appeler à la place?

- Le champ du code client affiche son message via `InvalidMessage`. Que devrait savoir le libellé récapitulatif sur ce champ pour le lister lui aussi, et qu'est-ce que cela vous apprend sur les cas où un provider partagé est le meilleur choix?
- Un utilisateur saisit une adresse e-mail bien formée qui appartient déjà à un autre contact. S'agit-il d'un retour sur un champ ou sur une règle métier, quand la vérification doit-elle s'exécuter, et où le message doit-il apparaître?

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 :

- ✓ Le gestionnaire `txtName_Validating` d'un collègue appelle `errorProvider.Clear()` dès que le nom devient valide. Qu'est-ce qui ne va pas pour un utilisateur qui a aussi saisi une adresse e-mail invalide, et qu'est-ce que le gestionnaire devrait appeler à la place?
- ✓ Le champ du code client affiche son message via `InvalidMessage`. Que devrait savoir le libellé récapitulatif sur ce champ pour le lister lui aussi, et qu'est-ce que cela vous apprend sur les cas où un provider partagé est le meilleur choix?
- ✓ Un utilisateur saisit une adresse e-mail bien formée qui appartient déjà à un autre contact. S'agit-il d'un retour sur un champ ou sur une règle métier, quand la vérification doit-elle s'exécuter, et où le message doit-il apparaître?

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 2 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 2 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Validation in Wisej.NET Course/Module 2`.



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-02.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.