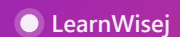


Exercice de code avec l'IA

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.



Briefez-le, lisez-le, puis cassez-le.

Cet exercice comporte trois parties, et seule la première concerne le prompt. Vous briefez un assistant avec la spécification du module et récupérez une solution ValidationClinic ; vous lisez ce qu'il a écrit et décidez s'il fait vraiment ce que le module demandait ; puis vous le poussez plus loin que ce que le module demandait. C'est dans la deuxième et la troisième partie que l'on apprend.

Ce que vous demandez

Remplacez les gestionnaires de champ écrits à la main dans ValidationClinic par l'extender Validation. Ajoutez un composant Validation et un ErrorProvider au formulaire d'inscription, définissez ContainerControl et validation.ErrorProvider, puis appelez SetValidationRules dans une méthode ConfigureValidationRules : Required puis Email sur txtEmail, Required puis Integer sur txtAge, Required puis Telephone sur txtPhone, Currency sur txtCreditLimit et une règle Regex pour le code client. Placez Required en premier partout pour que le message le plus utile l'emporte, abonnez-vous à l'événement Validating de l'extender pour rafraîchir le récapitulatif, et déplacez les tableaux de règles dans un petit utilitaire réutilisable pour qu'un autre formulaire puisse appliquer la même configuration.

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

- ✓ Composant Validation relié au formulaire et à une cible d'affichage ErrorProvider
- ✓ Méthode ConfigureValidationRules appelant SetValidationRules pour l'e-mail, l'âge, le téléphone, la limite de crédit et le code client
- ✓ Règle Required placée en premier sur chaque champ obligatoire, avec un texte InvalidMessage en langage simple
- ✓ Gestionnaire Validating de l'extender qui garde le récapitulatif de validation synchronisé avec les règles
- ✓ Utilitaire de configuration des règles réutilisable qu'un second formulaire peut appeler

Pourquoi il est conçu ainsi

À la fin du Module 2, le formulaire d'inscription de ValidationClinic fonctionne, mais regardez la quantité de code nécessaire: un utilitaire pour le nom, un pour l'adresse e-mail, un pour le téléphone, et chacun répète les mêmes vérifications « est-ce vide, est-ce la bonne forme ». Ajoutez l'âge, la limite de crédit et un

code client, et le formulaire s'alourdit d'une nouvelle poignée de méthodes presque identiques. Les vérifications ne sont pas fausses; elles sont simplement retapées pour chaque champ et chaque formulaire. Ce module remplace cette répétition par l'extender Validation, où vous décrivez à quoi ressemble un champ valide et où Wisej.NET exécute les vérifications pour vous.

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
Extender Validation	Un composant non visuel que vous ajoutez à un formulaire; il s'attache aux événements Validating et Validated de chaque contrôle qui a des règles, si bien que vous n'écrivez plus un gestionnaire par champ.
propriété étendue	Une propriété qu'un composant ajoute à d'autres contrôles; dès que l'extender est sur le formulaire, chaque contrôle compatible affiche une entrée ValidationRules dans la fenêtre Propriétés.
SetValidationRules	L'équivalent en code du designer: validation.SetValidationRules(txtEmail, new ValidationRule[] { ... }) attribue à un contrôle sa liste de règles à l'exécution.
règles intégrées	Des vérifications prêtes à l'emploi pour les saisies courantes: RequiredValidationRule, EmailValidationRule, IntegerValidationRule, DecimalValidationRule, CurrencyValidationRule, TelephoneValidationRule et RegexValidationRule.
ordre des règles	Les règles s'exécutent dans l'ordre où vous les listez; la première qui échoue arrête les suivantes et annule l'événement Validating du contrôle.
cible d'affichage	L'endroit où apparaît l' InvalidMessage de la règle en échec: sur le contrôle lui-même, via une icône ErrorProvider ou sous forme de texte dans un Label, le tout atteint par validation.ErrorProvider.
utilitaire de configuration des règles	Une petite méthode qui construit les tableaux de règles des champs courants, pour qu'un second formulaire obtienne les mêmes vérifications et la même formulation en l'appelant.

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 3. Vous pouvez ouvrir vous-même [module-03.md](#) pour voir exactement ce qu'il lira.



Utiliser ChatGPT

Ouvre [chatgpt.com](#) avec le prompt de ce module déjà chargé. Fonctionne avec GPT-



Utiliser Claude

Ouvre [claude.ai](#) avec le même prompt déjà chargé. Si vous utilisez Claude Code,

5 ou Codex ; toute offre capable de naviguer sur le web récupérera la spécification.

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 3 - « Règles de validation intégrées et étendre Validation » - 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-03.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 : Remplacez les gestionnaires de champ écrits à la main dans ValidationClinic par l'extender Validation. Ajoutez un composant Validation et un ErrorProvider au formulaire d'inscription, définissez `ContainerControl` et `validation.ErrorProvider`, puis appelez `SetValidationRules` dans une méthode `ConfigureValidationRules` : Required puis Email sur `txtEmail`, Required puis Integer sur `txtAge`, Required puis Telephone sur `txtPhone`, Currency sur `txtCreditLimit` et une règle Regex pour le code client. Placez Required en premier partout pour que le message le plus utile l'emporte, abonnez-vous à l'événement `Validating` de l'extender pour rafraîchir le récapitulatif, et déplacez les tableaux de règles dans un petit utilitaire réutilisable pour qu'un autre formulaire puisse appliquer la même configuration.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - Composant Validation relié au formulaire et à une cible d'affichage ErrorProvider
 - Méthode `ConfigureValidationRules` appelant `SetValidationRules` pour l'e-mail, l'âge, le téléphone, la limite de crédit et le code client
 - Règle Required placée en premier sur chaque champ obligatoire, avec un texte `InvalidMessage` en langage simple
 - Gestionnaire `Validating` de l'extender qui garde le récapitulatif de validation synchronisé avec les règles
 - Utilitaire de configuration des règles réutilisable qu'un second formulaire peut appeler
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 :

- Vous inversez l'ordre pour que la règle Email s'exécute avant Required sur `txtEmail`. Décrivez précisément ce que voit un utilisateur après avoir quitté le champ vide, et

pourquoi ce message est moins utile.

- Les utilitaires du Module 2 et l'extender aboutissent tous deux à une icône à côté du champ. Qu'avez-vous cessé d'écrire à la main, et que deviez-vous encore décider vous-même?
- Un collègue veut ajouter « la date de fin doit être postérieure à la date de début » à ClinicRules. Pourquoi l'extender n'est-il pas le bon endroit pour cette vérification, et quand ne s'exécuterait-elle pas?

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 :

- ✓ Vous inversez l'ordre pour que la règle Email s'exécute avant Required sur txtEmail. Décrivez précisément ce que voit un utilisateur après avoir quitté le champ vide, et pourquoi ce message est moins utile.
- ✓ Les utilitaires du Module 2 et l'extender aboutissent tous deux à une icône à côté du champ. Qu'avez-vous cessé d'écrire à la main, et que deviez-vous encore décider vous-même?
- ✓ Un collègue veut ajouter « la date de fin doit être postérieure à la date de début » à ClinicRules. Pourquoi l'extender n'est-il pas le bon endroit pour cette vérification, et quand ne s'exécuterait-elle pas?

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 3 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 3 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Validation in Wisej.NET Course/Module 3`.



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.

- ✓ **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-03.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."*



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.