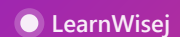


Exercice de code avec l'IA

Construisez l'exemple SupportDesk 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, ErrorProvider et retours utilisateur ». 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

Ajoutez la validation à l'éditeur de tickets du Support Desk : des DataAnnotations sur TicketEditModel, un TicketValidator qui renvoie des erreurs par champ et un récapitulatif, un ErrorProvider effacé avant chaque exécution et renseigné pour Title, Customer, Category et DueDate, un libellé récapitulatif pour les règles entre champs, Save désactivé tant que le modèle est invalide, un gestionnaire de DbUpdateException qui affiche un message compréhensible pour un numéro de ticket en double, et au moins cinq cas de test négatifs.

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

- ✓ DataAnnotations sur TicketEditModel et un TicketValidator qui renvoie des erreurs par champ et un récapitulatif
- ✓ ErrorProvider effacé avant chaque exécution et renseigné pour Title, Customer, Category et DueDate
- ✓ Libellé récapitulatif pour les règles entre champs et Save désactivé tant que le modèle est invalide
- ✓ DbUpdateException gérée avec un message compréhensible pour un numéro de ticket en double
- ✓ Au moins cinq cas de test négatifs

Pourquoi il est conçu ainsi

L'éditeur de tickets du Module 4 enregistre tout ce que l'utilisateur a saisi. Un titre vide, un ticket sans client ou un ticket fermé dont l'échéance tombe la semaine prochaine passent tous par SaveChangesAsync et ne sont arrêtés, au mieux, que par la base de données, qui répond par une exception du fournisseur

qu'un agent du support ne peut pas lire. Les utilisateurs ont besoin de messages immédiats et compréhensibles avant que la base ne rejette un enregistrement, et l'application a besoin d'une dernière ligne de défense qui reste en place quand l'interface est contournée par un import, un second écran ou un bug.

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
trois niveaux de validation	La validation de l'interface donne un retour immédiat, la validation du domaine impose les règles qui doivent être vraies partout, et les contraintes de la base garantissent la cohérence finale; chaque niveau intercepte ce que le précédent a laissé passer.
DataAnnotations sur le modèle d'édition	Des attributs comme [Required] et [StringLength] sur TicketEditModel décrivent les règles de saisie en un seul endroit déclaratif, séparé de l'entité EF Core.
validation explicite	Validator.TryValidateObject exécute les annotations à la demande; EF Core ne les exécute pas pour vous avant SaveChangesAsync, donc le formulaire doit l'appeler.
TicketValidator et ValidationMessage	Un service qui renvoie des éléments ValidationMessage(FieldName, Message), pour que l'ensemble de règles soit testable et que le formulaire se contente d'associer les messages aux contrôles.
ErrorProvider	L'extender Wisej.NET qui affiche une icône et une info-bulle à côté du contrôle invalide via SetError, et qui peut aussi être lié aux erreurs de données de l'élément courant.
récapitulatif de validation	Un libellé ou un MessageBox qui liste les erreurs entre champs n'appartenant à aucun contrôle en particulier, comme un ticket fermé avec une échéance future.
DbUpdateException	L'exception levée par EF Core quand la base rejette la modification; c'est le dernier filet de sécurité, et elle doit être traduite en message compréhensible, jamais affichée brute.

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,

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 SupportDesk 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, ErrorProvider et retours utilisateur » - du cours LearnWisej « Liaison de données avec EF Core », 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/ef-core-binding/specs/module-05.md>

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

1. Projet « SupportDesk », 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 la validation à l'éditeur de tickets du Support Desk : des `DataAnnotations` sur `TicketEditModel`, un `TicketValidator` qui renvoie des erreurs par champ et un récapitulatif, un `ErrorProvider` effacé avant chaque exécution et renseigné pour `Title`, `Customer`, `Category` et `DueDate`, un libellé récapitulatif pour les règles entre champs, `Save` désactivé tant que le modèle est invalide, un gestionnaire de `DbUpdateException` qui affiche un message compréhensible pour un numéro de ticket en double, et au moins cinq cas de test négatifs.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - `DataAnnotations` sur `TicketEditModel` et un `TicketValidator` qui renvoie des erreurs par champ et un récapitulatif
 - `ErrorProvider` effacé avant chaque exécution et renseigné pour `Title`, `Customer`, `Category` et `DueDate`
 - Libellé récapitulatif pour les règles entre champs et `Save` désactivé tant que le modèle est invalide
 - `DbUpdateException` gérée avec un message compréhensible pour un numéro de ticket en double
 - Au moins cinq cas de test négatifs
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 :

- Si vous retiriez toutes les `DataAnnotations` de `TicketEditModel` en les gardant sur l'entité `Ticket`, lesquels de vos cinq tests négatifs échoueraient encore avant `SaveChangesAsync`, et pourquoi?
- La règle des tickets fermés signale `DueDate` comme champ. Qu'est-ce qui changerait pour l'utilisateur si elle ne signalait aucun champ, et quand serait-ce le meilleur choix?

– Deux agents créent un ticket à la même seconde et le second enregistrement lève `DbUpdateException`. Que voit le second agent, que consigne le serveur, et lequel des deux montreriez-vous à un client?

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 :

- ✓ Si vous retiriez toutes les `DataAnnotations` de `TicketEditModel` en les gardant sur l'entité `Ticket`, lesquels de vos cinq tests négatifs échoueraient encore avant `SaveChangesAsync`, et pourquoi?
- ✓ La règle des tickets fermés signale `DueDate` comme champ. Qu'est-ce qui changerait pour l'utilisateur si elle ne signalait aucun champ, et quand serait-ce le meilleur choix?
- ✓ Deux agents créent un ticket à la même seconde et le second enregistrement lève `DbUpdateException`. Que voit le second agent, que consigne le serveur, et lequel des deux montreriez-vous à un client?

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 `Data Binding with EF Core 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.

- ✓ **Faites-le résister aux mauvais jours.** *"Examine ce projet SupportDesk 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 SupportDesk 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/ef-core-binding/specs/module-05.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.