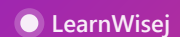


## Exercice de code avec l'IA

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.



### 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 SupportDesk ; 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

Construisez le navigateur de tickets du Support Desk : les records TicketListItem et TicketSearchCriteria, une méthode SearchTicketsAsync qui compose AsNoTracking, Where, OrderBy, Skip, Take et Select avant ToListAsync, un DataGridView lié à TicketListItem via une BindingSource, des ComboBox de référence pour le statut et le client chargées avant la première recherche, les boutons Search, Next Page et Previous Page, l'affichage du nombre total et de la taille de page, et une protection de chargement qui garde l'interface utilisable après une erreur.

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

- ✓ Records TicketListItem et TicketSearchCriteria avec SearchTicketsAsync
- ✓ DataGridView lié à TicketListItem via une BindingSource
- ✓ Filtrage et pagination composés dans la requête pour que la base de données fasse le travail
- ✓ ComboBox de référence chargées avant la première recherche, qui affichent des noms et stockent des clés
- ✓ Boutons Search, Next Page et Previous Page avec un nombre total et une protection de chargement

## Pourquoi il est conçu ainsi

Les opérateurs du Support Desk ont besoin d'une grille de tickets qui recherche rapidement sans charger toute la base dans la mémoire du serveur. Dans une application Wisej.NET, chaque session conserve sur le serveur ses propres formulaires, BindingSources et lignes sélectionnées; un écran qui charge des milliers d'entités Ticket suivies par utilisateur multiplie donc ce coût par le nombre de navigateurs ouverts. Ce module construit correctement le navigateur de tickets: composer la requête, l'exécuter de façon

asynchrone, matérialiser une petite page projetée et remettre cette liste à l'interface via une BindingSource.

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                                                                                                                                                                                                         |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| méthode d'exécution asynchrone  | ToListAsync, CountAsync et SingleOrDefaultAsync sont les appels qui envoient réellement le SQL; les attendre laisse le thread du serveur libre pendant que la base travaille.                                                 |
| projection                      | Un Select vers un record TicketListItem renvoie exactement les colonnes affichées par la grille, au lieu de l'entité Ticket avec son graphe Customer, Agent et Comments.                                                      |
| requête sans suivi              | AsNoTracking indique au Context de ne pas prendre de snapshot des résultats: moins de mémoire, aucune mise à jour accidentelle et une liste indépendante de toute unité de travail d'édition.                                 |
| BindingSource                   | Le composant Wisej.NET placé entre une collection de données et ses contrôles liés; il gère l'élément courant, la navigation et les opérations de liste, et garde la grille et les contrôles de détail synchronisés.          |
| IQueryable                      | La recette d'une requête de base de données, pas une liste stable: elle doit être exécutée et matérialisée avant que quoi que ce soit ne soit affecté à l'interface.                                                          |
| protection de chargement        | Un indicateur, des boutons désactivés et un libellé d'état autour de chaque appel attendu à la base, rétablis dans finally, pour qu'une seule opération s'exécute à la fois et que la page reste utilisable après une erreur. |
| liaison des listes de référence | Une ComboBox avec DisplayMember pour le nom affiché et ValueMember pour la clé stockée, chargée avant la première recherche ou l'affectation du modèle d'édition.                                                             |

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



### 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.



### **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 3 - « Charger les données dans BindingSource, DataGridView et les listes de référence » - 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-03.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 : Construisez le navigateur de tickets du Support Desk : les records TicketListItem et TicketSearchCriteria, une méthode SearchTicketsAsync qui compose AsNoTracking, Where, OrderBy, Skip, Take et Select avantToListAsync, un DataGridView lié à TicketListItem via une BindingSource, des ComboBox de référence pour le statut et le client chargées avant la première recherche, les boutons Search, Next Page et Previous Page, l'affichage du nombre total et de la taille de page, et une protection de chargement qui garde l'interface utilisable après une erreur.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
  - Records TicketListItem et TicketSearchCriteria avec SearchTicketsAsync
  - DataGridView lié à TicketListItem via une BindingSource
  - Filtrage et pagination composés dans la requête pour que la base de données fasse le travail
  - ComboBox de référence chargées avant la première recherche, qui affichent des noms et stockent des clés
  - Boutons Search, Next Page et Previous Page avec un nombre total et une protection de chargement
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 :

- Quelle durée de vie avez-vous choisie pour le DbContext dans SearchTicketsAsync, et qu'est-ce qui ne fonctionnerait plus si la page gardait un Context vivant pendant toute sa durée de vie?
- Après une recherche, quelles données sont conservées sur le serveur pour cette session et lesquelles restent uniquement dans la base? Que coûte un second opérateur simultané?
- Si la liste de référence des clients atteignait cinquante mille lignes, quelles parties de

la page changeriez-vous et quelles parties du service de requête resteraient exactement identiques?

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 :

- ✓ Quelle durée de vie avez-vous choisie pour le DbContext dans SearchTicketsAsync, et qu'est-ce qui ne fonctionnerait plus si la page gardait un Context vivant pendant toute sa durée de vie?
- ✓ Après une recherche, quelles données sont conservées sur le serveur pour cette session et lesquelles restent uniquement dans la base? Que coûte un second opérateur simultané?
- ✓ Si la liste de référence des clients atteignait cinquante mille lignes, quelles parties de la page changeriez-vous et quelles parties du service de requête resteraient exactement identiques?

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 `Data Binding with EF Core 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.

- ✓ **Relisez-le comme une pull request.** *"Fais-moi parcourir ce code SupportDesk 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 SupportDesk 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 gèrerait pas."*
- ✓ **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."*



### 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.