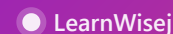


Exercice de code avec l'IA

Construisez l'exemple OpsMonitor 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.



Vous ne taperez pas cet exemple. Vous le piloterez.

Presque plus personne n'écrit du code Wisej.NET à partir d'un fichier vide — un assistant rédige le premier jet et le développeur décide s'il est correct. Cela ne fonctionne que si vous maîtrisez le sujet, et c'est ce que le reste de ce module vous a apporté. Ici, vous confiez ces connaissances à ChatGPT ou à Claude, vous récupérez l'exemple du module sous forme de solution Visual Studio fonctionnelle, et vous le relisez comme vous reliriez la pull request d'un collègue.

Ce que vous demandez

Profilez les deux scénarios dominés par l'attente dans WisejPerfLab. Exécutez la recherche de tickets sous l'outil Database et associez chaque requête à l'action d'interface qui l'a produite : le chargement de la grille émet une requête client par ligne affichée et la requête de détail ramène de grandes colonnes de texte que personne n'affiche. Remplacez-les par une seule requête de projection sans suivi qui ne sélectionne que les colonnes de la grille et pagine le résultat. Exécutez ensuite l'export CSV sous File I/O et .NET Async, repérez l'appel bloquant .Result dans le gestionnaire de clic et les petites écritures répétées, et déplacez la génération dans un traitement en arrière-plan Application.StartTask qui écrit le fichier en flux et signale une progression limitée au lieu de pousser chaque étape.

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

- ✓ Trace Database associant les requêtes aux actions d'interface, avec le nombre de requêtes N+1 avant la correction
- ✓ Requête unique de projection sans suivi avec pagination, remplaçant les requêtes par ligne et celles qui ramènent trop de données
- ✓ Preuves File I/O et .NET Async pour l'export, identifiant l'appel bloquant et le schéma d'écriture
- ✓ Export déplacé dans une tâche en arrière-plan qui écrit en flux et ne pousse la progression qu'aux étapes significatives
- ✓ Nombre de requêtes, durée des requêtes et temps réel de l'export avant/après, avec les risques restants

Pourquoi il est conçu ainsi

Le Module 2 a mesuré 1 840 ms de temps écoulé et n'a trouvé que 300 ms de CPU. Le Module 3 a supprimé l'essentiel de ce CPU et l'écran n'est toujours pas rapide, car les 600 ms manquantes n'ont jamais

été du calcul — c'était de l'attente. C'est la catégorie la plus sous-diagnostiquée d'une application métier: le serveur est au repos, la courbe CPU est plate, et l'utilisateur regarde un indicateur de chargement. Quelque part, une requête s'exécute une fois par ligne, un formulaire de détail ramène une colonne que personne n'affiche, ou un gestionnaire de clic bloque sur.Result pendant qu'un thread du pool ne fait rien. Les outils qui voient cela sont Database, File I/O,.NET Async et Instrumentation, et c'est dans ce module que la latence restante de WisejPerfLab reçoit enfin un nom.

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
outil Database	Enregistre les requêtes ADO.NET et EF Core émises par un scénario, avec leur durée et leur ordre, pour que chaque requête puisse être associée à l'action d'interface qui l'a provoquée.
requête N+1	Une requête pour lister, puis une de plus par élément renvoyé; la liste paraît correcte avec dix lignes et s'effondre à mille, et c'est invisible pour le profilage CPU.
surcharge de données (over-fetching)	Sélectionner des colonnes ou des lignes que l'écran n'affiche jamais — les grandes colonnes de texte et binaires sont les coupables habituels, et elles coûtent du temps de requête, du transfert et de la mémoire.
requête sans suivi	Une requête en lecture seule qui ne met pas en place le suivi des modifications, adaptée aux grilles et aux listes qui affichent des données plutôt qu'elles ne les modifient.
outil.NET Async	Montre les activités asynchrones et l'endroit où elles attendent: c'est ainsi qu'une chaîne involontairement séquentielle ou bloquée devient visible.
blocage sur de l'asynchrone	Appeler.Result ou.Wait() depuis un gestionnaire; cela monopolise un thread pendant l'attente et, sous charge, produit des symptômes qui ressemblent à une famine du pool de threads.
export en arrière-plan	Une génération longue de fichier ou de rapport sortie du chemin immédiat du clic avec Application.StartTask, qui signale sa progression à des intervalles significatifs plutôt qu'à chaque étape.

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 6. Vous pouvez ouvrir vous-même [module-06.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 OpsMonitor 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 6 - « Base de données, E/S fichier, asynchrone et attentes externes » - du cours LearnWisej « Performances et profilage », 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/performance-and-profiling/specs/module-06.md>

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

1. Projet « OpsMonitor », 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 : Profilez les deux scénarios dominés par l'attente dans `WisejPerfLab`. Exécutez la recherche de tickets sous l'outil Database et associez chaque requête à l'action d'interface qui l'a produite : le chargement de la grille émet une requête client par ligne affichée et la requête de détail ramène de grandes colonnes de texte que personne n'affiche. Remplacez-les par une seule requête de projection sans suivi qui ne sélectionne que les colonnes de la grille et pagine le résultat. Exécutez ensuite l'export CSV sous File I/O et .NET Async, repérez l'appel bloquant `.Result` dans le gestionnaire de clic et les petites écritures répétées, et déplacez la génération dans un traitement en arrière-plan `Application.StartTask` qui écrit le fichier en flux et signale une progression limitée au lieu de pousser chaque étape.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - Trace Database associant les requêtes aux actions d'interface, avec le nombre de requêtes N+1 avant la correction
 - Requête unique de projection sans suivi avec pagination, remplaçant les requêtes par ligne et celles qui ramènent trop de données
 - Preuves File I/O et .NET Async pour l'export, identifiant l'appel bloquant et le schéma d'écriture
 - Export déplacé dans une tâche en arrière-plan qui écrit en flux et ne pousse la progression qu'aux étapes significatives
 - Nombre de requêtes, durée des requêtes et temps réel de l'export avant/après, avec les risques restants
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 :

- La recherche émet désormais une requête au lieu de deux cent une, mais le temps écoulé n'a

été divisé que par deux. Où se trouve le reste du temps, et quel outil utiliseriez-vous pour le découvrir?

- Votre collègue affirme que l'export ne pose pas de problème, car.Result « ne bloque qu'un thread ». Expliquez ce qui se passe avec cinquante exports simultanés et quels écrans l'utilisateur verrait ralentir.

- Parmi les éléments que ce lab met en cache, lesquels peuvent être partagés sans risque entre les sessions et lesquels ne le peuvent pas, et quelle règle avez-vous appliquée pour en décider?

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 :

- ✓ La recherche émet désormais une requête au lieu de deux cent une, mais le temps écoulé n'a été divisé que par deux. Où se trouve le reste du temps, et quel outil utiliseriez-vous pour le découvrir?
- ✓ Votre collègue affirme que l'export ne pose pas de problème, car.Result « ne bloque qu'un thread ». Expliquez ce qui se passe avec cinquante exports simultanés et quels écrans l'utilisateur verrait ralentir.
- ✓ Parmi les éléments que ce lab met en cache, lesquels peuvent être partagés sans risque entre les sessions et lesquels ne le peuvent pas, et quelle règle avez-vous appliquée pour en décider?

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 6 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 6 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Performance and Profiling Course/Module 6`.



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 OpsMonitor 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/performance-and-profiling/specs/module-06.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 OpsMonitor 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.