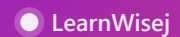


Exercice de code avec l'IA

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.



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 OpsMonitor ; 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

Corrigez le bouton Refresh de WisejPerfLab. Profilez le scénario d'actualisation avec CPU Usage, ouvrez le flame graph et Show Hot Path, et identifiez les deux coupables : un formateur qui s'exécute pour chaque cellule à chaque actualisation et une reconstruction de contrôles qui recrée le panneau des KPI à chaque fois. Remplacez-les par un view model DashboardSnapshot dont les chaînes d'affichage sont calculées une seule fois dans le service, modifiez les labels et la source de données du graphique une seule fois dans une portée de sonde plutôt que ligne par ligne, et gardez le bouton désactivé dans un try/finally pendant le traitement. Réexécutez le même scénario et capturez le nouveau chemin critique.

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

- ✓ Trace CPU Usage « avant » identifiant le formateur répété et la reconstruction de contrôles avec leur CPU total et propre
- ✓ View model DashboardSnapshot avec des chaînes d'affichage précalculées dans le service
- ✓ Gestionnaire d'actualisation groupé qui modifie les contrôles une seule fois dans une portée de sonde, avec réactivation dans un try/finally
- ✓ Trace CPU Usage « après » du même scénario avec le nouveau chemin critique
- ✓ Tableau avant/après du CPU total, de la fonction principale et du nombre d'appels, avec le risque restant

Pourquoi il est conçu ainsi

Le Module 2 s'est terminé sur un nom: TicketFormatter.FormatRow, 68,9 % de CPU propre, appelée cinquante mille fois en un seul clic. C'est la forme de presque tous les problèmes de CPU dans une interface côté serveur. Personne n'a écrit une méthode lente; quelqu'un a écrit une méthode raisonnable, puis s'est arrangé pour qu'elle s'exécute une fois par cellule, à chaque actualisation, indéfiniment. Le second coupable du tableau de bord de WisejPerfLab en est le reflet inversé: RebuildKpiPanel n'est

appelée qu’une fois, mais elle libère et recrée tout un panneau de contrôles qui n’avaient pas besoin de changer. L’un est un problème de nombre d’appels, l’autre un problème de travail par appel, et ce module vous apprend à reconnaître lequel vous avez et à le corriger sans deviner.

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
coût d’un gestionnaire d’événements	Un clic, un changement de sélection ou un tick de timer s’exécute sur le serveur et bloque l’événement suivant de cette session: la quantité de logique déclenchée par le clic correspond au temps pendant lequel l’utilisateur est figé.
flame graph	Les mêmes données d’arbre des appels dessinées sous forme de barres de largeur proportionnelle, ce qui rend immédiatement visible une frame large isolée — une fonction qui dévore le scénario.
CPU propre ou nombre d’appels	Un CPU propre élevé avec peu d’appels signifie que le corps est coûteux; un coût par appel faible avec un nombre d’appels énorme signifie qu’il faut l’appeler moins souvent, pas le micro-optimiser.
view model précalculé	Un petit objet contenant des chaînes et des valeurs prêtes à afficher, construit une seule fois par un service, pour que la couche d’interface affecte des valeurs au lieu de les formater.
modification groupée de l’interface	Définir en une seule passe, avec des valeurs préparées, les propriétés que l’utilisateur doit voir, au lieu de modifier des contrôles dans une boucle qui effectue aussi du travail.
reconstruction de contrôles	Libérer et recréer des contrôles pour les actualiser; cela relance la construction, la mise en page et l’application du thème, et c’est presque toujours plus coûteux que de mettre à jour les contrôles existants.
cache de données de référence immuables	Des données de référence partagées par toute l’application et jamais modifiées; on peut les mettre en cache globalement sans risque, contrairement à l’état propre à chaque utilisateur, qui doit rester dans la session.

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



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 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 3 - « Chemins critiques CPU, gestionnaires d'événements et travail d'interface côté serveur » - 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-03.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 : Corrigez le bouton Refresh de `WisejPerfLab`. Profilez le scénario d'actualisation avec CPU Usage, ouvrez le flame graph et Show Hot Path, et identifiez les deux coupables : un formateur qui s'exécute pour chaque cellule à chaque actualisation et une reconstruction de contrôles qui recrée le panneau des KPI à chaque fois. Remplacez-les par un view model `DashboardSnapshot` dont les chaînes d'affichage sont calculées une seule fois dans le service, modifiez les labels et la source de données du graphique une seule fois dans une portée de sonde plutôt que ligne par ligne, et gardez le bouton désactivé dans un `try/finally` pendant le traitement. Réexécutez le même scénario et capturez le nouveau chemin critique.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - Trace CPU Usage « avant » identifiant le formateur répété et la reconstruction de contrôles avec leur CPU total et propre
 - View model `DashboardSnapshot` avec des chaînes d'affichage précalculées dans le service
 - Gestionnaire d'actualisation groupé qui modifie les contrôles une seule fois dans une portée de sonde, avec réactivation dans un `try/finally`
 - Trace CPU Usage « après » du même scénario avec le nouveau chemin critique
 - Tableau avant/après du CPU total, de la fonction principale et du nombre d'appels, avec le risque restant
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 :

- Les deux coupables étaient lents, mais pour des raisons opposées. Expliquez lequel relevait d'un problème de nombre d'appels et lequel d'un problème de travail par appel, et pourquoi les preuves du profileur diffèrent pour chacun.
- Un collègue propose de marquer le gestionnaire d'actualisation `async` pour le rendre plus

rapide. Qu'est-ce qui changerait réellement, qu'est-ce qui ne changerait pas, et comment la trace le montrerait-elle?

– Votre snapshot est construit une fois par actualisation. Quel risque d'obsolescence ou de correction cela introduit-il, et que faudrait-il changer si le même snapshot était mis en cache entre les sessions?

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 :

- ✓ Les deux coupables étaient lents, mais pour des raisons opposées. Expliquez lequel relevait d'un problème de nombre d'appels et lequel d'un problème de travail par appel, et pourquoi les preuves du profileur diffèrent pour chacun.
- ✓ Un collègue propose de marquer le gestionnaire d'actualisation async pour le rendre plus rapide. Qu'est-ce qui changerait réellement, qu'est-ce qui ne changerait pas, et comment la trace le montrerait-elle?
- ✓ Votre snapshot est construit une fois par actualisation. Quel risque d'obsolescence ou de correction cela introduit-il, et que faudrait-il changer si le même snapshot était mis en cache entre les sessions?

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 `Performance and Profiling 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 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."*
- ✓ **Remplacez une pièce sans déplacer les jointures.** *"Choisis la partie de ce projet OpsMonitor 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 gèrait pas."*
- ✓ **Faites-le résister aux mauvais jours.** *"Examine ce projet OpsMonitor 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.