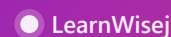


Exercice de code avec l'IA

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

Créez la solution WisejPerfLab avec un tableau de bord de support, une grille de tickets et une arborescence de clients alimentés par des données de départ, puis rendez-la mesurable. Ajoutez un service ScenarioProbe qui encadre un scénario dans une portée Stopwatch et écrit des logs PERF structurés de début et de fin avec le nom du scénario, l'action de l'utilisateur, les millisecondes écoulées et le nombre de lignes, enregistrez-le dans le host builder et encadrez-y trois scénarios : l'actualisation du tableau de bord, la recherche de tickets et le déploiement d'un nœud de l'arborescence des clients. Exécutez chaque scénario une fois pour la mise en température, puis enregistrez la référence et rédigez un budget de performance qui précise l'environnement, la configuration de build, la taille du jeu de données et un seuil d'acceptation par scénario.

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

- ✓ Solution WisejPerfLab avec un tableau de bord, une grille de tickets et une arborescence de clients sur des données de départ
- ✓ Service ScenarioProbe écrivant des logs PERF structurés de début et de fin avec les millisecondes écoulées et le nombre de lignes
- ✓ Trois scénarios instrumentés: actualisation du tableau de bord, recherche de tickets et déploiement de l'arborescence des clients
- ✓ Note de référence consignait l'environnement, la configuration de build, la taille du jeu de données et la procédure de mise en température
- ✓ Budget de performance avec un seuil d'acceptation par scénario et l'outil qui permettrait de prouver chacun

Pourquoi il est conçu ainsi

Une équipe de support signale que WisejPerfLab « est devenu lent ». Cette phrase ne contient aucune mesure, aucun scénario et aucune couche. Le tableau de bord attend peut-être SQL, le gestionnaire d'actualisation formate peut-être cinquante mille cellules, le navigateur redessine peut-être une grille qui n'aurait jamais dû être rendue en entier, ou le serveur retient tout simplement les formulaires de deux cents sessions fermés il y a une heure. Chacun de ces cas appelle une correction différente et un outil Visual Studio différent, et le développeur qui ouvre le profileur avant d'avoir décidé duquel il s'agit collectera une trace incapable de répondre à la question. Ce module construit d'abord le modèle mental: ce qui se passe réellement entre un clic dans le navigateur et un écran redessiné, ce que coûte chacune de ces étapes, et comment consigner un scénario et un budget pour que les six modules suivants disposent d'un point de comparaison.

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
interface côté serveur	Les contrôles, leurs propriétés et leur état résident dans le processus serveur et le navigateur affiche ce que le serveur décrit: chaque objet d'interface que vous créez est donc de la mémoire serveur, et chaque modification de l'interface est du CPU serveur plus une charge utile.
aller-retour d'un événement	Le chemin complet d'une action utilisateur: événement dans le navigateur, transport vers le serveur, votre gestionnaire, accès aux données, modifications des contrôles, mise à jour sérialisée en retour, nouveau rendu dans le navigateur — l'unité que l'utilisateur ressent réellement.
la session comme unité de capacité	Une session possède des pages, des formulaires, des contrôles, des timers, des sélections, des bindings et des données en cache; les questions de capacité se traitent par session, et non par requête, car cette mémoire reste résidente entre les clics.
push WebSocket	Le canal par lequel le serveur envoie des mises à jour que l'utilisateur n'a pas demandées; il rend peu coûteuse la diffusion de données en direct et rend coûteuse une tâche en arrière-plan non limitée, car chaque état intermédiaire devient une frame.
les cinq catégories	Le CPU dans votre code, le blocage et l'attente, l'allocation et la rétention, l'accès aux données et les E/S, et la charge utile des mises à jour — la classification qui détermine quel outil du profileur peut même voir votre problème.
latence ou débit	La latence est le temps d'attente d'un utilisateur; le débit est le nombre de sessions que le serveur tient. Une correction qui supprime une attente améliore la latence; une correction qui supprime du travail ou de la mémoire améliore les deux, et certaines corrections de latence se paient en débit.
scénario et référence	Une séquence nommée et reproductible d'actions utilisateur, avec l'environnement, la configuration de build, la taille du jeu de données et les chiffres mesurés qu'elle a produits — sans cela, une trace n'est que du bruit et toute comparaison ultérieure est dénuée de sens.

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 1. Vous pouvez ouvrir vous-même [module-01.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 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 1 – « Le modèle de performance de Wisej.NET » – 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-01.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 : Créez la solution WisejPerfLab avec un tableau de bord de support, une grille de tickets et une arborescence de clients alimentés par des données de départ, puis rendez-la mesurable. Ajoutez un service ScenarioProbe qui encadre un scénario dans une portée Stopwatch et écrit des logs PERF structurés de début et de fin avec le nom du scénario, l'action de l'utilisateur, les millisecondes écoulées et le nombre de lignes, enregistrez-le dans le host builder et encadrez-y trois scénarios : l'actualisation du tableau de bord, la recherche de tickets et le déploiement d'un nœud de l'arborescence des clients. Exécutez chaque scénario une fois pour la mise en température, puis enregistrez la référence et rédigez un budget de performance qui précise l'environnement, la configuration de build, la taille du jeu de données et un seuil d'acceptation par scénario.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - Solution WisejPerfLab avec un tableau de bord, une grille de tickets et une arborescence de clients sur des données de départ
 - Service ScenarioProbe écrivant des logs PERF structurés de début et de fin avec les millisecondes écoulées et le nombre de lignes
 - Trois scénarios instrumentés: actualisation du tableau de bord, recherche de tickets et déploiement de l'arborescence des clients
 - Note de référence consignant l'environnement, la configuration de build, la taille du jeu de données et la procédure de mise en température
 - Budget de performance avec un seuil d'acceptation par scénario et l'outil qui permettrait de prouver chacun
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 :

- L'actualisation de votre tableau de bord journalise 1 900 ms, mais le CPU du serveur bouge à peine pendant le scénario. Quelle catégorie cela exclut-il, vers laquelle cela pointe-t-il, et quelle est la mesure suivante la moins coûteuse?
- La sonde mesure le gestionnaire. Citez deux coûts que l'utilisateur subit sur le même clic et que votre chiffre elapsedMs n'inclut pas, et indiquez comment vous mesureriez chacun.
- Pourquoi la note de référence doit-elle consigner la taille du jeu de données et la configuration de build, alors que le nom du scénario et le temps écoulé semblent être la partie intéressante?

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 :

- ✓ L'actualisation de votre tableau de bord journalise 1 900 ms, mais le CPU du serveur bouge à peine pendant le scénario. Quelle catégorie cela exclut-il, vers laquelle cela pointe-t-il, et quelle est la mesure suivante la moins coûteuse?
- ✓ La sonde mesure le gestionnaire. Citez deux coûts que l'utilisateur subit sur le même clic et que votre chiffre elapsedMs n'inclut pas, et indiquez comment vous mesureriez chacun.
- ✓ Pourquoi la note de référence doit-elle consigner la taille du jeu de données et la configuration de build, alors que le nom du scénario et le temps écoulé semblent être la partie intéressante?

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 1 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 1 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Performance and Profiling Course/Module 1`.



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-01.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.