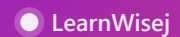


Exercice de code avec l'IA

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.



L'assistant, ce sont les mains rapides. L'architecte, c'est toujours vous.

Un assistant peut produire l'exemple de ce module en quelques minutes. Ce qu'il ne peut pas faire, c'est décider si le résultat est la bonne conception pour votre application — cette décision vous revient. C'est vous qui apportez le jugement : briefez-le correctement, puis vérifiez ce qu'il vous renvoie à la lumière de ce que le Module 2 vient de vous apprendre.

Ce que vous demandez

Profilez correctement l'actualisation lente du tableau de bord de WisejPerfLab. Passez en Release, faites chauffer l'application une fois, puis collectez avec Alt+F2 une trace CPU Usage de ce seul scénario, restreignez la chronologie récapitulative au clic lui-même et utilisez Show Hot Path pour trouver le chemin principal sous les frames de dispatch de Wisej.NET. Collectez une seconde trace Instrumentation du même scénario pour obtenir le nombre d'appels et le temps réel écoulé, déterminez à partir des deux traces si le coût est du CPU dans votre code ou du temps passé à attendre, et enregistrez les deux fichiers .diagsession sous un nom qui porte le module, le scénario, le build, le jeu de données et l'horodatage.

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

- ✓ Trace CPU Usage en Release du scénario d'actualisation du tableau de bord, chronologie restreinte au clic
- ✓ Capture d'écran du chemin critique indiquant la principale fonction applicative et son CPU propre par rapport à son CPU total
- ✓ Trace Instrumentation du même scénario avec le nombre d'appels et le temps réel écoulé
- ✓ Note de trace indiquant le cours, le module, le scénario, le build, l'hébergement, l'outil, le navigateur et le budget attendu
- ✓ Paragraphe sur la cause racine soupçonnée, indiquant vers laquelle des cinq catégories de coût pointent les éléments de preuve

Pourquoi il est conçu ainsi

Le Module 1 a laissé WisejPerfLab avec un chiffre et aucune explication: l'actualisation du tableau de bord prend 1 840 ms. L'étape suivante évidente consiste à la profiler, et c'est précisément là que la plupart des investigations dérapent. Une trace collectée sous le débogueur mesure le débogueur. Une trace démarrée avant la mise en température de l'application mesure le JIT. Une trace qui couvre deux minutes de clics

mesure tout et ne prouve rien. Une trace collectée avec quatre outils activés à la fois mesure la surcharge du profileur lui-même. Ce module est la discipline mécanique qui fait d'une trace une preuve recevable: le bon build, le bon outil pour la catégorie soupçonnée, le bon processus quand l'application n'est pas lancée par Visual Studio, un seul scénario par collecte, et un rapport que vous savez lire.

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
Performance Profiler (Alt+F2)	La voie de collecte a posteriori: il peut lancer le projet sans débogueur, collecter un scénario complet et produire un rapport enregistré, et c'est de là que viennent les vraies preuves.
fenêtre Diagnostic Tools	La vue en direct, attachée au débogueur; utile pour un coup d'œil rapide pendant l'exécution pas à pas, trompeuse comme source de mesures de temps, car le débogueur modifie le comportement du runtime.
matrice de choix des outils	CPU Usage pour le calcul, Instrumentation pour les nombres d'appels et le temps réel, Object Allocation pour le renouvellement de la mémoire, Memory Usage pour la rétention, Database pour les requêtes, File I/O pour les fichiers,.NET Async pour les chaînes d'await,.NET Counters pour la santé en direct.
.diagsession	Le fichier de trace enregistré; le nommer avec le module, le scénario, le build, le jeu de données et l'horodatage est ce qui rend possible une comparaison avant/après des semaines plus tard.
chronologie récapitulative	Le graphique CPU ou mémoire sur toute la collecte; restreindre la sélection à l'action exacte de l'utilisateur est l'étape qui retire le démarrage et la mise en température de l'arbre des appels.
chemin critique (hot path)	La chaîne d'appels la plus coûteuse de la plage sélectionnée; Show Hot Path la déploie pour que vous puissiez trouver votre propre code sous les frames du framework.
CPU propre ou total	Le total inclut les appels enfants, le CPU propre ne concerne que le corps de la fonction: un CPU propre élevé signifie que la méthode elle-même est coûteuse, un total élevé avec un CPU propre faible signifie que le coût se trouve dans ce qu'elle appelle.

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 2. Vous pouvez ouvrir vous-même [module-02.md](#) pour voir exactement ce qu'il lira.



Utiliser ChatGPT

Ouvre [chatgpt.com](#) avec le prompt de ce module déjà chargé. Fonctionne avec GPT-



Utiliser Claude

Ouvre [claude.ai](#) avec le même prompt déjà chargé. Si vous utilisez Claude Code,

5 ou Codex ; toute offre capable de 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 2 - « Démarche de profilage Visual Studio pour 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-02.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 correctement l'actualisation lente du tableau de bord de WisejPerfLab. Passez en Release, faites chauffer l'application une fois, puis collectez avec Alt+F2 une trace CPU Usage de ce seul scénario, restreignez la chronologie récapitulative au clic lui-même et utilisez Show Hot Path pour trouver le chemin principal sous les frames de dispatch de Wisej.NET. Collectez une seconde trace Instrumentation du même scénario pour obtenir le nombre d'appels et le temps réel écoulé, déterminez à partir des deux traces si le coût est du CPU dans votre code ou du temps passé à attendre, et enregistrez les deux fichiers .diagsession sous un nom qui porte le module, le scénario, le build, le jeu de données et l'horodatage.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - Trace CPU Usage en Release du scénario d'actualisation du tableau de bord, chronologie restreinte au clic
 - Capture d'écran du chemin critique indiquant la principale fonction applicative et son CPU propre par rapport à son CPU total
 - Trace Instrumentation du même scénario avec le nombre d'appels et le temps réel écoulé
 - Note de trace indiquant le cours, le module, le scénario, le build, l'hébergement, l'outil, le navigateur et le budget attendu
 - Paragraphe sur la cause racine soupçonnée, indiquant vers laquelle des cinq catégories de coût pointent les éléments de preuve
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 :

- Votre trace CPU Usage ne représente que 300 ms des 1 840 ms mesurées par la sonde. Que signifie cet écart, et quel outil utiliseriez-vous pour la collecte suivante?
- Une méthode du framework se trouve en haut du chemin critique avec un CPU total très élevé

et presque aucun CPU propre. Expliquez ce que cela signifie et ce que vous examineriez au lieu de signaler un bug du framework.

– Pourquoi une autre personne a-t-elle besoin de la note de trace pour reproduire votre résultat, alors que le.diagsession contient déjà toute la trace?

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 :

- ✓ Votre trace CPU Usage ne représente que 300 ms des 1 840 ms mesurées par la sonde. Que signifie cet écart, et quel outil utiliseriez-vous pour la collecte suivante?
- ✓ Une méthode du framework se trouve en haut du chemin critique avec un CPU total très élevé et presque aucun CPU propre. Expliquez ce que cela signifie et ce que vous examineriez au lieu de signaler un bug du framework.
- ✓ Pourquoi une autre personne a-t-elle besoin de la note de trace pour reproduire votre résultat, alors que le.diagsession contient déjà toute la trace?

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 2 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 2 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Performance and Profiling Course/Module 2`.



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.

- ✓ **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-02.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."*
- ✓ **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 gérait pas."*



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.