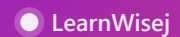


Exercice de code avec l'IA

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 7 vient de vous apprendre.

Ce que vous demandez

Réalisez le projet de synthèse WisejPerfLab. Établissez un modèle de capacité à partir de vos propres mesures : mémoire retenue par session inactive, CPU par scénario courant, simultanéité attendue au pic et marge de sécurité que vous conservez. Rédigez un HealthCheck.json dont les valeurs maxSessions, maxMemory et maxCPU découlent de ces chiffres, en expliquant chaque seuil, et qui renvoie 503 avec un retry-after pour qu'un répartiteur de charge oriente les nouveaux utilisateurs ailleurs pendant que les sessions existantes continuent de fonctionner. Rédigez les notes de déploiement pour deux instances derrière un répartiteur de charge avec affinité de session et prise en charge de WebSocket, puis assemblez le rapport final : scénarios, environnement, preuves de référence, causes racines, modifications, preuves « après », risques restants et compteurs de production qui signaleront le retour d'une régression.

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

- ✓ Modèle de capacité déduisant le nombre de sessions par serveur de la mémoire mesurée par session et du CPU par scénario
- ✓ HealthCheck.json avec maxSessions, maxMemory, maxCPU, code de retour et retry-after, chaque seuil étant justifié
- ✓ Notes de déploiement couvrant l'affinité de session, la prise en charge de WebSocket et le repli sur le polling
- ✓ Rapport de performance final avec référence, cause racine, modification et preuve « après » pour chaque correction de module
- ✓ Plan de supervision indiquant les compteurs, les seuils et les alertes qui détecteraient chaque régression

Pourquoi il est conçu ainsi

Six modules de profilage ont produit un WisejPerfLab plus rapide et un dossier de traces. Rien de tout cela ne répond à la question que l'équipe d'exploitation posera le jour du déploiement: combien d'utilisateurs tiennent sur un serveur, et que se passe-t-il quand il est plein? Le profilage sur une machine de développement vous donne le coût d'un scénario; la production a besoin que ce coût soit converti en modèle de capacité, en seuils qui refusent les nouvelles sessions avant que le processus ne soit en difficulté, en architecture de déploiement qui respecte l'état de session côté serveur, et en supervision qui remarquera si l'une des six corrections régresse. Cette traduction est l'objet de ce module, et le rapport de synthèse est ce qui rend le travail vérifiable par quelqu'un qui n'a jamais ouvert Visual Studio.

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
modèle de capacité	La mémoire retenue par session inactive et le CPU par scénario courant, multipliés par la simultanéité attendue au pic et rapportés aux ressources de l'instance, avec une marge conservée délibérément.
HealthCheck.json	La configuration par laquelle un serveur Wisej.NET peut se déclarer indisponible en fonction du nombre de sessions, de la mémoire et du CPU, pour qu'un répartiteur de charge cesse de lui envoyer de nouveaux utilisateurs.
503 avec retry-after	La réponse qu'une instance en mauvaise santé renvoie à une première requête; les sessions existantes continuent de tourner sur leur nœud pendant que les nouveaux utilisateurs sont orientés ailleurs.
affinité de session	L'acheminement systématique d'un client vers l'instance qui détient l'état de sa session, généralement mis en œuvre avec des cookies d'affinité; sans elle, une session Wisej.NET ne peut pas survivre à une deuxième requête.
équilibrage compatible WebSocket	Une configuration du répartiteur de charge qui laisse passer la connexion mise à niveau; quand WebSocket n'est pas disponible, l'application se replie sur le polling et le push en temps réel est perdu.
compteurs de production	Des métriques continues et légères — CPU, tas GC, taux d'allocation, taux d'exceptions, état du pool de threads — qui révèlent une régression sans profileur attaché.
rapport de performance final	Scénarios, environnement, preuves de référence, cause racine, modification, preuves « après », risques restants, seuils et plan de supervision, réunis dans un seul document vérifiable.

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 7. Vous pouvez ouvrir vous-même [module-07.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 7 - « Montée en charge, health checks, équilibrage de charge et optimisation finale » - 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-07.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 : Réalisez le projet de synthèse WisejPerfLab. Établissez un modèle de capacité à partir de vos propres mesures : mémoire retenue par session inactive, CPU par scénario courant, simultanéité attendue au pic et marge de sécurité que vous conservez. Rédigez un HealthCheck.json dont les valeurs maxSessions, maxMemory et maxCPU découlent de ces chiffres, en expliquant chaque seuil, et qui renvoie 503 avec un retry-after pour qu'un répartiteur de charge oriente les nouveaux utilisateurs ailleurs pendant que les sessions existantes continuent de fonctionner. Rédigez les notes de déploiement pour deux instances derrière un répartiteur de charge avec affinité de session et prise en charge de WebSocket, puis assemblez le rapport final : scénarios, environnement, preuves de référence, causes racines, modifications, preuves « après », risques restants et compteurs de production qui signaleront le retour d'une régression.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - Modèle de capacité déduisant le nombre de sessions par serveur de la mémoire mesurée par session et du CPU par scénario
 - HealthCheck.json avec maxSessions, maxMemory, maxCPU, code de retour et retry-after, chaque seuil étant justifié
 - Notes de déploiement couvrant l'affinité de session, la prise en charge de WebSocket et le repli sur le polling
 - Rapport de performance final avec référence, cause racine, modification et preuve « après » pour chaque correction de module
 - Plan de supervision indiquant les compteurs, les seuils et les alertes qui détecteraient chaque régression
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 health check refuse les nouvelles sessions à 150. Un collègue demande pourquoi pas 300, puisque le processus n'a pas planté à 300 pendant les tests. Répondez avec vos propres chiffres.
- Un déploiement est mis en production derrière un répartiteur de charge sans affinité de session. Décrivez exactement ce que vit l'utilisateur, et pourquoi il s'agit d'une défaillance fonctionnelle plutôt que d'un ralentissement.
- Dans six mois, l'actualisation du tableau de bord est de nouveau lente. Quel signal de production vous alerterait en premier, et que collecteriez-vous ensuite?

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 health check refuse les nouvelles sessions à 150. Un collègue demande pourquoi pas 300, puisque le processus n'a pas planté à 300 pendant les tests. Répondez avec vos propres chiffres.
- ✓ Un déploiement est mis en production derrière un répartiteur de charge sans affinité de session. Décrivez exactement ce que vit l'utilisateur, et pourquoi il s'agit d'une défaillance fonctionnelle plutôt que d'un ralentissement.
- ✓ Dans six mois, l'actualisation du tableau de bord est de nouveau lente. Quel signal de production vous alerterait en premier, et que collecteriez-vous ensuite?

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 7 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 7 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Performance and Profiling Course/Module 7`.



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-07.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.