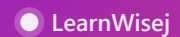


Exercice de code avec l'IA

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.



Écrire le code, c'est désormais la moitié facile.

Obtenir d'un assistant moderne du code qui fonctionne pour « Mémoire, allocations et fuites de session » ne demande qu'un bon prompt. Le juger, le corriger et l'étendre demande tout ce que ce module a couvert. C'est sur ce partage que repose l'exercice : vous apportez la spécification et la relecture, l'assistant apporte la frappe.

Ce que vous demandez

Attaquez les deux moitiés du problème de mémoire dans WisejPerfLab. Exécutez le scénario de recherche de tickets sous .NET Object Allocation et repérez les modèles de ligne et les chaînes alloués à chaque recherche, puis réduisez-les en projetant une seule fois au lieu de reconstruire des objets à chaque rafraîchissement de l'affichage. Prenez ensuite un instantané Memory Usage après la mise en température, ouvrez et fermez cinquante fois le formulaire de détail d'un ticket, prenez un second instantané et comparez : les formulaires retenus sont enracinés par un abonnement statique à GlobalTicketBus.TicketChanged et par un timer d'actualisation jamais arrêté. Désabonnez l'un et arrêtez l'autre dans une surcharge de Dispose, videz la source de données du BindingSource, puis réexécutez le scénario d'ouverture/fermeture et montrez que le tas retenu revient à un niveau proche de l'instantané A.

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

- ✓ Trace d'allocation identifiant le type qui alloue le plus et le chemin d'appel pour le scénario de recherche, avant et après
- ✓ Comparaison des instantanés A/B montrant le nombre d'instances retenues du formulaire de détail avant la correction
- ✓ Capture d'écran du chemin vers la racine identifiant l'abonnement à l'événement statique et le timer actif
- ✓ Surcharge de Dispose qui désabonne l'événement statique, arrête et libère le timer et vide le binding
- ✓ Budget mémoire par session et checklist de libération, avec l'instantané « après » prouvant que le chemin de rétention a disparu

Pourquoi il est conçu ainsi

Le tableau de bord de WisejPerfLab respecte désormais son budget CPU, et pourtant l'application s'effondre au bout de quatre heures en production. Les problèmes de mémoire ne se comportent pas comme les problèmes de CPU: ils sont invisibles dans une démo de deux minutes, ils ne se manifestent jamais sous la forme d'un clic lent, et ce sont eux qui expliquent qu'un serveur qui tient confortablement quatre-vingts sessions le matin refuse des connexions en milieu d'après-midi. Le lab en contient deux, et ce n'est pas le même problème. La recherche de tickets alloue bien plus que nécessaire, ce qui coûte du temps de GC et du CPU pendant son exécution. Le formulaire de détail d'un ticket n'est jamais réellement libéré après sa fermeture par l'utilisateur, ce qui coûte de la mémoire pour toujours. Le premier se trouve avec une trace d'allocation, le second uniquement avec deux instantanés et un chemin vers la racine.

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
pression d'allocation	La quantité de mémoire qu'un scénario crée pendant son exécution; une forte pression augmente la fréquence du GC et vole du CPU, même quand rien ne fuit.
mémoire retenue	Ce qui reste accessible alors que le scénario devrait être terminé; c'est ce qui se multiplie par le nombre de sessions et décide de la capacité.
.NET Object Allocation	L'outil qui enregistre chaque allocation managée pendant un scénario et les regroupe par type et par chemin d'appel qui alloue.
instantanés Memory Usage	Des captures du tas prises à des moments choisis; comparer l'instantané A et l'instantané B montre quels types ont gagné des instances jamais libérées.
chemin vers la racine	La chaîne de références qui maintient un objet en vie; c'est la preuve concrète d'une fuite, et le seul moyen de prouver que la fuite a disparu.
racine de rétention	Ce qui se trouve en haut de cette chaîne — un champ statique, un événement statique, un timer actif, une tâche en arrière-plan ou un cache à longue durée de vie.
budget mémoire par session	Un chiffre, en mégaoctets retenus par session inactive, qui transforme « l'application consomme beaucoup de mémoire » en verdict de réussite ou d'échec, puis en seuil de health check.

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 4. Vous pouvez ouvrir vous-même [module-04.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 4 - « Mémoire, allocations et fuites de session » - 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-04.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 : Attaquez les deux moitiés du problème de mémoire dans `WisejPerfLab`. Exécutez le scénario de recherche de tickets sous .NET Object Allocation et repérez les modèles de ligne et les chaînes alloués à chaque recherche, puis réduisez-les en projetant une seule fois au lieu de reconstruire des objets à chaque rafraîchissement de l'affichage. Prenez ensuite un instantané Memory Usage après la mise en température, ouvrez et fermez cinquante fois le formulaire de détail d'un ticket, prenez un second instantané et comparez : les formulaires retenus sont enracinés par un abonnement statique à `GlobalTicketBus.TicketChanged` et par un timer d'actualisation jamais arrêté. Désabonnez l'un et arrêtez l'autre dans une surcharge de `Dispose`, videz la source de données du `BindingSource`, puis réexécutez le scénario d'ouverture/fermeture et montrez que le tas retenu revient à un niveau proche de l'instantané A.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - Trace d'allocation identifiant le type qui alloue le plus et le chemin d'appel pour le scénario de recherche, avant et après
 - Comparaison des instantanés A/B montrant le nombre d'instances retenues du formulaire de détail avant la correction
 - Capture d'écran du chemin vers la racine identifiant l'abonnement à l'événement statique et le timer actif
 - Surcharge de `Dispose` qui désabonne l'événement statique, arrête et libère le timer et vide le binding
 - Budget mémoire par session et checklist de libération, avec l'instantané « après » prouvant que le chemin de rétention a disparu
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 d'allocation s'est nettement améliorée, mais la comparaison d'instantanés est inchangée. Quel type de problème avez-vous corrigé, lequel ne l'avez-vous pas corrigé, et comment un utilisateur percevrait-il la différence?

- Pourquoi le lab ouvre-t-il et ferme-t-il le formulaire cinquante fois plutôt qu'une, et que concluriez-vous d'une seule instance présente dans l'instantané B?

- Le callback du timer tient en une ligne et ne fait que définir un label. Expliquez précisément comment il maintient en vie tout un formulaire, et ce qu'une vérification `IsDisposed` résout et ne résout pas.

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 d'allocation s'est nettement améliorée, mais la comparaison d'instantanés est inchangée. Quel type de problème avez-vous corrigé, lequel ne l'avez-vous pas corrigé, et comment un utilisateur percevrait-il la différence?
- ✓ Pourquoi le lab ouvre-t-il et ferme-t-il le formulaire cinquante fois plutôt qu'une, et que concluriez-vous d'une seule instance présente dans l'instantané B?
- ✓ Le callback du timer tient en une ligne et ne fait que définir un label. Expliquez précisément comment il maintient en vie tout un formulaire, et ce qu'une vérification `IsDisposed` résout et ne résout pas.

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 4 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 4 sur GitHub](#)

Ou clonez l'ensemble avec `git clone https://github.com/iceteagroup/LearnWisej-Samples` et ouvrez `Performance and Profiling Course/Module 4`.



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.

- ✓ **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."*
- ✓ **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."*
- ✓ **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."*



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.