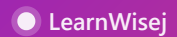


Exercice de code avec l'IA

Construisez l'exemple TicketOpsLive 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 le premier écran de TicketOps Live : une page avec un bandeau d'état en direct, un label d'horloge, un indicateur de progression et un heartbeat serveur simulé qui met à jour le navigateur sans aucun clic de l'utilisateur. Critères d'acceptation : L'interface se met à jour une fois par seconde après que l'apprenant a cliqué sur Start ; Le navigateur affiche les modifications sans clic supplémentaire ; Stop désactive la boucle en moins d'une seconde ; Démarrer deux fois ne crée pas deux boucles concurrentes ; Le code vérifie si la page a été libérée (disposed).

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

- ✓ Ajoutez un panneau supérieur nommé statusPanel.
- ✓ Ajoutez les labels clockLabel, connectionLabel et activityLabel.
- ✓ Ajoutez une barre de progression nommée serverLoadBar.
- ✓ Ajoutez les boutons startButton et stopButton.
- ✓ Implémentez une tâche en arrière-plan qui simule le heartbeat lorsque l'on clique sur startButton.
- ✓ Le heartbeat doit mettre à jour l'horloge, une valeur de charge fictive et un message d'activité une fois par seconde.
- ✓ Arrêtez les mises à jour lorsque l'on clique sur stopButton ou lorsque la page est libérée (disposed).

Pourquoi il est conçu ainsi

On associe généralement les applications web temps réel au chat, aux jeux, aux cotations boursières, aux écrans de supervision et aux tableaux de bord. Dans les applications métier, le même besoin apparaît sous

des formes moins spectaculaires mais plus précieuses: un superviseur voit arriver les nouveaux tickets, un opérateur de production voit une machine s'arrêter, un répartiteur voit qu'une intervention a été réaffectée, ou un employé de la facturation voit qu'un import est toujours en cours sans appuyer sur Actualiser.

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 TicketOpsLive 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 – « Architecture web temps réel dans Wisej.NET » – du cours LearnWisej « Applications temps réel avec push serveur », 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/realtime-server-push/specs/module-01.md>

Construis ensuite un projet Wisej.NET 4 complet que je puisse ouvrir dans Visual Studio :

1. Projet « TicketOpsLive », 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 le premier écran de TicketOps Live : une page avec un bandeau d'état en direct, un label d'horloge, un indicateur de progression et un heartbeat serveur simulé qui met à jour le navigateur sans aucun clic de l'utilisateur. Critères d'acceptation : L'interface se met à jour une fois par seconde après que l'apprenant a cliqué sur Start ; Le navigateur affiche les modifications sans clic supplémentaire ; Stop désactive la boucle en moins d'une seconde ; Démarrer deux fois ne crée pas deux boucles concurrentes ; Le code vérifie si la page a été libérée (disposed).
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
 - Ajoutez un panneau supérieur nommé `statusPanel`
 - Ajoutez les labels `clockLabel`, `connectionLabel` et `activityLabel`
 - Ajoutez une barre de progression nommée `serverLoadBar`
 - Ajoutez les boutons `startButton` et `stopButton`
 - Implémentez une tâche en arrière-plan qui simule le heartbeat lorsque l'on clique sur `startButton`
 - Le heartbeat doit mettre à jour l'horloge, une valeur de charge fictive et un message d'activité une fois par seconde
 - Arrêtez les mises à jour lorsque l'on clique sur `stopButton` ou lorsque la page est libérée (disposed)
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 :

- Pourquoi ce lab appelle-t-il `Application.Update(this)` à l'intérieur de la boucle?
- Qu'est-ce qui change si `WebSocket` n'est pas disponible?
- Pourquoi `_running` n'est-il pas un champ statique?
- Quel serait le risque d'une mise à jour toutes les 10 millisecondes?

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 :

- ✓ Pourquoi ce lab appelle-t-il `Application.Update(this)` à l'intérieur de la boucle?
- ✓ Qu'est-ce qui change si `WebSocket` n'est pas disponible?
- ✓ Pourquoi `_running` n'est-il pas un champ statique?
- ✓ Quel serait le risque d'une mise à jour toutes les 10 millisecondes?

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 `Real-Time Server Push 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 TicketOpsLive 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/realtime-server-push/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 TicketOpsLive 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.