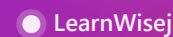


# Développement assisté par l'IA, documentation prête pour MCP et livraison du projet de synthèse

Guide de la leçon · Guide du lab / de l'examen



## Guide de la leçon

Discipline du code assisté par l'IA, accès à la documentation, prompts de revue, soutenance finale en contexte d'entreprise. Ce que cela signifie pour une équipe Wisej.NET avancée — et comment aborder ce module.



### Enjeu de production

Discipline du code assisté par l'IA, accès à la documentation, prompts de revue, soutenance finale en contexte d'entreprise.

## 1. Le problème en production

Les équipes avancées utiliseront des outils d'IA, mais le travail Wisej.NET en entreprise exige toujours de l'architecture, une documentation ancrée dans les sources, de la revue de code, de la revue de sécurité et une responsabilité opérationnelle.

## 2. Concepts clés

| Concept                         | Pourquoi c'est important                                                                                                                                                                     |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| développement assisté par l'IA  | Utiliser des outils d'IA pour rédiger des brouillons de code, de tests et de documentation, tandis que l'équipe reste responsable de la revue, de l'ancrage dans les sources et du résultat. |
| prompts ancrés dans les sources | Prompts qui orientent l'assistant vers le code, la documentation et les conventions réels du projet, pour que le résultat généré corresponde à l'architecture réelle.                        |
| endpoint de documentation MCP   | Serveur Model Context Protocol qui expose la documentation de l'équipe aux outils d'IA, pour que les réponses proviennent des propres sources du projet.                                     |
| checklist de revue              | Liste fixe de vérifications d'architecture, de sécurité et de conventions appliquée à chaque modification générée par l'IA avant son acceptation.                                            |
| soutenance de l'architecture    | Présenter et justifier, preuves à l'appui, les décisions de conception du projet de synthèse, pour que l'équipe puisse les challenger et consigner le résultat.                              |
| démo du projet de synthèse      | Démonstration de bout en bout de l'application terminée, prouvant que les schémas des modules fonctionnent ensemble dans des conditions réalistes.                                           |

### 3. Déroulé de la leçon

#### L'IA aide l'équipe à aller plus vite

L'IA peut rédiger des écrans, des services, des tests, des prompts, de la documentation et des inventaires de migration. Elle ne peut pas assumer le risque en production. Le développeur avancé doit fournir le contexte d'architecture, vérifier les API, relire le code généré et tester les chemins d'échec.

#### Utiliser la documentation comme source pour les outils

La documentation de Wisej.NET peut être considérée comme la source de vérité du comportement actuel de la plateforme. Les prompts d'IA doivent demander du code qui respecte les conventions du projet et exiger des citations ou des références lorsqu'ils utilisent des API propres à la plateforme.

#### De la vitesse sans code non relu

L'assistance de l'IA apporte le plus de valeur en périphérie du travail : rédiger un squelette de service à partir d'une définition de commande, générer des cas de test pour une matrice des chemins d'échec, écrire un inventaire de migration à partir d'un fichier de solution ou transformer un runbook en checklist. Dans chaque cas, le développeur sait déjà à quoi ressemble un résultat correct et peut vérifier la sortie rapidement. Elle apporte le moins de valeur là où la vérification est difficile, comme la logique de sécurité, la concurrence ou les API de plateforme que le développeur connaît mal.

La règle appliquée par le lab est que le code généré est un brouillon provenant d'un contributeur inconnu. Il passe par la même revue que n'importe quelle pull request, avec les mêmes questions : où vit l'état, qu'est-ce qui peut échouer, qu'est-ce qui est sécurisé et qu'est-ce qui change en production. Le chemin d'échec du lab montre pourquoi. Un service généré invente une API Wisej.NET qui n'existe pas et stocke des données par utilisateur dans un champ static. Les deux compileraient lors d'une démo rapide et les deux échoueraient avec de vraies sessions.

N'acceptez une suggestion que si quelqu'un de l'équipe peut l'expliquer sans l'outil.

#### Des prompts et des checklists ancrés dans le projet

La qualité du code généré dépend de la qualité du contexte fourni. Chaque session d'IA du lab commence par un en-tête de prompt qui énonce les règles du projet : la structure de la solution, l'emplacement de la logique métier, le fait que les statics ne contiennent jamais de données de session, que les services appliquent les permissions et que seules des API Wisej.NET documentées peuvent être utilisées, avec une référence pour chacune. La bibliothèque de prompts rassemble ces en-têtes et les prompts propres à chaque tâche pour que toute l'équipe parte de la même base.

La checklist de revue du code généré en est l'autre moitié. Elle est assez courte pour être appliquée à chaque modification et assez précise pour détecter les défaillances enseignées dans le cours : un champ static contenant un état d'utilisateur ou de tenant, un gestionnaire d'UI contenant de la logique de persistance, une API introuvable dans la documentation, un contrôle de permission absent d'un service, une exception remontée à l'utilisateur et une ressource jamais libérée.

- Le prompt nomme-t-il les conventions que le résultat doit respecter ?
- Chaque affirmation propre à la plateforme dans le résultat cite-t-elle la documentation ?
- La checklist a-t-elle été appliquée et signée par un relecteur, et non par l'auteur ?

## Une documentation lisible par les outils et par les mainteneurs

Un projet qui sera maintenu avec l'aide de l'IA a besoin d'une documentation qu'un outil peut récupérer, et pas seulement qu'une personne peut parcourir. Un endpoint de documentation MCP expose à l'assistant les notes d'architecture, les decision records, les conventions et les runbooks de l'équipe, pour que les réponses concernant ce projet proviennent des sources de ce projet plutôt que de données d'entraînement générales. Le même endpoint sert un nouveau membre de l'équipe qui pose les mêmes questions.

Cela ne fonctionne que si la documentation est structurée. L'index de la documentation du lab liste chaque document avec une ligne décrivant son objet, les décisions qu'il consigne et la date de sa dernière vérification. Les architecture decision records du Module 1 sont les entrées les plus précieuses, car ils expliquent pourquoi une conception existe et un outil peut les citer lorsqu'on lui demande de la modifier. Viennent ensuite les runbooks, la matrice des permissions, la matrice des chemins d'échec et le tableau de configuration.

Écrivez pour le lecteur qui n'a jamais rencontré l'équipe. Si une note repose sur des connaissances qui n'existent que dans la tête de quelqu'un, elle sera fausse dans un an.

## Défendre le projet de synthèse comme une revue d'architecture

Le projet de synthèse est livré sous la forme d'une revue d'architecture de production plutôt que d'une démo de fonctionnalités. Le script de démo guide un relecteur à travers EnterpriseOps Command Center dans un ordre fixe : la structure de la solution et les decision records, une session sensible au tenant et une requête inter-tenants rejetée, une commande qui traverse un service jusqu'à la base de données et revient avec son entrée d'audit, un import en arrière-plan avec progression et annulation, un workflow avec un échec compensé, la page de diagnostic, et le package de déploiement avec son retour arrière.

Chaque étape associe un écran fonctionnel à un chemin d'échec, car la revue porte sur ce qui se passe quand les choses tournent mal. Le relecteur demandera où vit l'état, qui peut appeler quoi, comment on annule une mauvaise release et ce que l'équipe ferait à deux heures du matin. La déclaration de préparation à la production y répond par écrit, liste ce qui n'est pas encore fait et nomme le responsable de chaque point ouvert.

Une soutenance réussit lorsque chaque décision peut être rattachée à une raison, que chaque risque a une mesure d'atténuation nommée et que l'équipe peut dire clairement ce qu'elle changerait ensuite.



### Réflexe de production avancé

Ne vous arrêtez pas quand l'écran paraît correct. Expliquez à qui appartient l'état, les implications de sécurité, les chemins d'échec et le comportement en production avant de considérer le module comme terminé.



### Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Utiliser l'assistance de l'IA pour accélérer le développement Wisej.NET sans accepter de code non relu.

- ✓ Créer des prompts et des checklists de revue fondés sur les standards du projet et la documentation officielle.
- ✓ Préparer des notes de projet prêtes pour MCP et la documentation à l'intention des futurs mainteneurs.
- ✓ Livrer et défendre le projet de synthèse avancé sous la forme d'une revue d'architecture de production.

## Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



### Lab pratique

Finalisez le dossier du projet de synthèse. Ajoutez des notes sur l'usage de l'IA, une bibliothèque de prompts, des liens vers la documentation, une checklist de revue de code, un script de démo final, le plan de la présentation de soutenance de l'architecture et une déclaration de préparation à la production.

## Approche suggérée

- ✓ Lisez le problème en production et identifiez le risque d'architecture avant de toucher à l'UI.
- ✓ Créez ou mettez à jour d'abord l'encapsulation par un service, de commande, de modèle ou de composant concernée.
- ✓ Construisez l'écran ou le composant Wisej.NET en suivant les standards de projet établis.
- ✓ Ajoutez une démonstration du chemin d'échec, pas seulement du chemin nominal.
- ✓ Consignez ce qui a changé dans le journal d'apprentissage et mettez à jour la checklist concernée.

## Livrables attendus

- ✓ Bibliothèque de prompts IA
- ✓ Checklist de revue du code généré
- ✓ Index de la documentation du projet
- ✓ Script de démo du projet de synthèse
- ✓ Déclaration de préparation à la production

## Questions de révision

- ✓ Quel code généré a été accepté, et pourquoi ?
- ✓ Chaque affirmation propre à la plateforme a-t-elle été vérifiée dans la documentation ?
- ✓ L'apprenant peut-il défendre les décisions de déploiement, de sécurité et de gestion de l'état ?



### Réflexe de production avancé

Ne vous arrêtez pas quand l'écran paraît correct. Expliquez à qui appartient l'état, les implications de sécurité, les chemins d'échec et le comportement en production avant de considérer le module comme terminé.