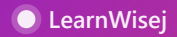


## Exercice de code avec l'IA

Construisez l'exemple IconDesk 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.



### **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 IconDesk.Icons comme bibliothèque de classes à côté de l'application, placez au moins douze icônes SVG dans un répertoire Resources et marquez-les Embedded Resource. Ajoutez un catalogue statique Applcons qui expose chaque icône sous forme de chaîne resource.wx nommée pour que les appelants n'aient jamais à saisir un chemin, référencez la bibliothèque depuis IconDesk, puis affectez certaines icônes via le designer et d'autres depuis le catalogue en C#. Vérifiez le suffixe de couleur sur trois icônes et notez le nettoyage SVG que certaines ont demandé. Terminez par les règles de nommage et de versionnage qui décident comment une nouvelle icône entre dans le pack.

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

- ✓ Bibliothèque de classes IconDesk.Icons avec au moins douze fichiers SVG dans un répertoire Resources, tous en Embedded Resource
- ✓ Catalogue statique Applcons exposant chaque icône sous forme de chaîne resource.wx nommée
- ✓ Page IconDesk affectant certaines icônes via le designer et d'autres depuis le catalogue Applcons
- ✓ Trois icônes vérifiées avec le suffixe de couleur, avec une note sur le nettoyage SVG qu'elles ont demandé
- ✓ Règles de nommage et de versionnage décrivant comment une nouvelle icône entre dans le pack et comment une application en remplace une

### Pourquoi il est conçu ainsi

Au cinquième module, IconDesk affiche un logo embarqué, une image de thème et des icônes issues de deux packs officiels, et c'est là que le désordre commence: des chaînes de ressources apparaissent en ligne dans les gestionnaires d'événements, deux écrans orthographient différemment le même fichier, et personne ne sait dire quelles icônes le produit possède. Un pack personnalisé est le remède économique,

et il achète une frontière: un assembly contient les visuels, un catalogue C# les nomme, et le reste de l'application ne saisit plus jamais un chemin.

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                                                                                                                                                                                                     |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| assembly de ressources               | Une simple bibliothèque de classes.NET dont l'unique rôle est de transporter des fichiers d'icônes en ressources embarquées; elle n'a besoin d'aucune référence à l'application ni de code Wisej.NET propre.              |
| la recette en quatre étapes          | Créez une bibliothèque de classes, créez un répertoire nommé Resources, placez-y les fichiers d'icônes, définissez leur Build Action sur Embedded Resource. Tout ce qui suit relève du confort.                           |
| catalogue d'icônes                   | Une classe statique telle qu' Applcons qui expose chaque icône sous forme de constante de chaîne nommée, si bien que le compilateur et l'IntelliSense remplacent les chemins copiés-collés.                               |
| nom sémantique contre nom de fichier | Le catalogue expose le vocabulaire de votre produit — enregistrer, supprimer, client, avertissement — pendant que le fichier garde un nom simple, si bien que le renommer change une ligne et non quarante écrans.        |
| adresse de ressource qualifiée       | Une icône dans un assembly distinct s'adresse par resource.wx/ plus le nom de l'assembly plus le chemin de la ressource, et c'est pourquoi toutes les entrées du catalogue partagent une seule constante racine privée.   |
| contrat de recolorage                | Le suffixe?color= ne recolore que les visuels monochromes adaptés; un SVG avec des valeurs fill codées en dur est la raison habituelle d'une icône qui refuse de changer de couleur.                                      |
| jeux de thèmes et remplacements      | Des packages parallèles partageant une même API de catalogue sur des visuels différents, plus des fichiers déployés qui priment sur une ressource embarquée, remplacent des icônes sans toucher au code de l'application. |

## 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 6. Vous pouvez ouvrir vous-même [module-06.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



### Utiliser Claude

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

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 IconDesk 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 6 – « Créer votre propre pack d'icônes Wisej.NET » – du cours LearnWisej « Icônes et images dans Wisej.NET », 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/icons-and-images/specs/module-06.md>

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

1. Projet « IconDesk », 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 `IconDesk.Icons` comme bibliothèque de classes à côté de l'application, placez au moins douze icônes SVG dans un répertoire `Resources` et marquez-les `Embedded Resource`. Ajoutez un catalogue statique `AppIcons` qui expose chaque icône sous forme de chaîne `resource.wx` nommée pour que les appelants n'aient jamais à saisir un chemin, référencez la bibliothèque depuis `IconDesk`, puis affectez certaines icônes via le designer et d'autres depuis le catalogue en C#. Vérifiez le suffixe de couleur sur trois icônes et notez le nettoyage SVG que certaines ont demandé. Terminez par les règles de nommage et de versionnage qui décident comment une nouvelle icône entre dans le pack.
3. Produis ces livrables et indique-moi où se trouve chacun d'eux dans la solution :
  - Bibliothèque de classes `IconDesk.Icons` avec au moins douze fichiers SVG dans un répertoire `Resources`, tous en `Embedded Resource`
  - Catalogue statique `AppIcons` exposant chaque icône sous forme de chaîne `resource.wx` nommée
  - Page `IconDesk` affectant certaines icônes via le designer et d'autres depuis le catalogue `AppIcons`
  - Trois icônes vérifiées avec le suffixe de couleur, avec une note sur le nettoyage SVG qu'elles ont demandé
  - Règles de nommage et de versionnage décrivant comment une nouvelle icône entre dans le pack et comment une application en remplace une
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 pack compile et le designer n'affiche rien dans l'Image Selector. Quelle étape de la recette vérifieriez-vous en premier, et quelle preuve vous dirait que les ressources sont

réellement dans l'assembly?

- Deux de vos trois icônes de test se recolorent et la troisième reste grise. Qu'y a-t-il dans ce SVG, et quand laisseriez-vous délibérément une icône non recolorable?
- Un client veut faire remplacer trois icônes dans son déploiement uniquement. Laquelle des voies de remplacement choisiriez-vous, et que vous coûte ce choix à la prochaine version du pack?

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 pack compile et le designer n'affiche rien dans l'Image Selector. Quelle étape de la recette vérifieriez-vous en premier, et quelle preuve vous dirait que les ressources sont réellement dans l'assembly?
- ✓ Deux de vos trois icônes de test se recolorent et la troisième reste grise. Qu'y a-t-il dans ce SVG, et quand laisseriez-vous délibérément une icône non recolorable?
- ✓ Un client veut faire remplacer trois icônes dans son déploiement uniquement. Laquelle des voies de remplacement choisiriez-vous, et que vous coûte ce choix à la prochaine version du pack?

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.



### Pas encore d'exemple de référence pour ce cours — le vôtre est le premier

Le [dépôt des exemples LearnWisej](#) contient des solutions complètes pour la plupart des cours, mais pas encore pour celui-ci. Il n'y a donc rien à quoi vous comparer et aucun raccourci : la spécification et votre propre relecture sont votre seul filet de sécurité. Gardez ce que vous construisez — et si le résultat est réussi, nous aimerions le voir.

## Trois prompts pour aller plus loin

Orientez votre assistant vers le projet que vous venez de construire 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/icons-and-images/specs/module-06.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 IconDesk 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 IconDesk 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 gèrait 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.