

AI Coding Exercise

Build this module's IconDesk sample with ChatGPT or Claude from a ready-made prompt that points the model at this module's specification, then review, run and extend what it gives you.

● LearnWisej



Writing the code is the easy half now.

Getting working code for "Wisej.NET Icon Packs and NuGet" out of a modern assistant takes one good prompt. Judging it, correcting it and extending it takes everything this module covered. That is the split this exercise is built around — you supply the specification and the review, the assistant supplies the typing.

What you are asking for

Add two official Wisej-4 icon-pack NuGet packages to IconDesk and build an IconCompare page that shows the same five concepts, Save, Delete, Search, User and Settings, drawn from both families so the visual difference is impossible to miss. Pick at least five of the icons through the Visual Studio image explorer rather than typing the path, assign the rest from the pack catalog in C#, recolor two of them with the ?color= suffix, and watch the network panel to see icon files arriving one request at a time and then coming from the browser cache. Close with a short recommendation naming the primary icon family for the project.

The module asks for these deliverables, and the prompt below asks the assistant for exactly them:

- ✓ Two official Wisej-4 icon-pack NuGet packages referenced by IconDesk
- ✓ IconCompare page showing Save, Delete, Search, User and Settings from both packs
- ✓ At least five icons chosen through the Visual Studio image explorer rather than typed by hand
- ✓ Two icons recoloured with the colour suffix and a note on which artwork accepted the recolour
- ✓ One-paragraph recommendation naming the project's primary icon family, with the browser-cache evidence behind it

Why it is shaped this way

Three modules in, IconDesk can already show a picture five different ways, but every icon it owns was drawn or downloaded by hand. A real application needs a Save, a Delete, a Search, a User and a Settings icon on day one, and another forty by the end of the quarter, all in the same visual language. Copying SVG files in one at a time produces a folder of unrelated artwork, inconsistent stroke weights and no licence trail.

These are the terms the assistant has to get right. If what comes back uses them loosely, it has not understood the module:

Term	What it means here
icon pack	An ordinary.NET assembly whose icon files, mostly SVG, are compiled in as embedded resources; referencing it adds one file to your deployment no matter how many icons it holds.
embedded resource	A file compiled into the assembly rather than copied beside it, so the icons cannot be lost, renamed or forgotten when the application is deployed to another server.
resource.wx	The Wisej.NET handler that serves one embedded resource over HTTP; an icon-pack image source is a resource.wx URL naming the pack assembly and the file inside it.
icon catalog	The static class a pack publishes so callers get IntelliSense and a compile-time name instead of a typed path; the exact namespace and class name vary from pack to pack.
per-icon request	Pack icons are fetched individually, on first use, and then served from the browser cache, which is what makes a catalog of thousands of icons affordable.
theme image preload	Theme images travel with the theme and are already in the client theme cache when the application starts, so they behave very differently from pack icons at startup.
primary icon family	The one pack a product commits to for its everyday commands, chosen deliberately for coverage, style and licence rather than accumulated pack by pack.

Build it — pick your assistant

Both buttons open a new tab with the **same ready-made prompt** already filled in. The prompt points the model at this module's specification — a single Markdown file holding the lesson, the lab brief and the deliverables — so it works from the real material instead of guessing what Module 4 is about. You can open [module-04.md](#) yourself to see exactly what it will read.



Use ChatGPT

Opens [chatgpt.com](#) with this module's prompt loaded. Works with GPT-5 or Codex; any plan that can browse the web will fetch the spec.



Use Claude

Opens [claude.ai](#) with the same prompt loaded. Claude Code users can paste it into a terminal in an empty folder instead.



Send it as-is and you get a project, not a snippet

The prompt ends by asking for the whole solution as a single [.zip](#) with build-and-run steps. Download it, open the solution in Visual Studio, press F5 and IconDesk comes up on [localhost](#). You need the .NET SDK and the [Wisej-4](#) NuGet package; everything else the prompt asks the assistant to write itself.

The prompt, in full — read it, then edit it if you want a different sample

I am working through Module 4 – "Wisej.NET Icon Packs and NuGet" – of the LearnWisej Icons and Images in Wisej.NET course, and I want to build the module sample with you.

Start by reading the module specification. It is a single Markdown file holding the full lesson, the lab brief and the deliverables:

<https://learnwisej.com/assets/courses/icons-and-images/specs/module-04.md>

Then build a complete Wisej.NET 4 project I can open in Visual Studio:

1. Project "IconDesk", C#, net8.0 or later, NuGet package Wisej-4 4.1.x, Kestrel host calling `app.UseWisej()`.
2. Build what the specification's lab asks for: Add two official Wisej-4 icon-pack NuGet packages to IconDesk and build an IconCompare page that shows the same five concepts, Save, Delete, Search, User and Settings, drawn from both families so the visual difference is impossible to miss. Pick at least five of the icons through the Visual Studio image explorer rather than typing the path, assign the rest from the pack catalog in C#, recolor two of them with the `?color=` suffix, and watch the network panel to see icon files arriving one request at a time and then coming from the browser cache. Close with a short recommendation naming the primary icon family for the project.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Two official Wisej-4 icon-pack NuGet packages referenced by IconDesk
 - IconCompare page showing Save, Delete, Search, User and Settings from both packs
 - At least five icons chosen through the Visual Studio image explorer rather than typed by hand
 - Two icons recoloured with the colour suffix and a note on which artwork accepted the recolour
 - One-paragraph recommendation naming the project's primary icon family, with the browser-cache evidence behind it
4. Follow the specification wherever it names a Wisej.NET class, property or event – use exactly that one. Where it leaves a choice open, tell me which API you picked and why.
5. Keep the server authoritative: state and validation live in .NET, and the browser renders and reports. Never move a decision into JavaScript because it is easier there.
6. Make the result demonstrable: I should be able to run it and see the behaviour this module is about, including what happens when something goes wrong.

Before you write any code, restate in your own words what you are going to build and where each piece will live, so I can correct the design while it is still cheap.

When the code is done, package the whole solution as a single .zip I can download, and give me the exact steps to build and run it.

Finally, answer these questions from the specification, using your own implementation as the evidence:

- Referencing a pack of three thousand icons adds one assembly to the deployment. What does that cost the browser, and when?
- One of your two recoloured icons ignored the colour suffix. What is it about that artwork, and what would you change in the pack selection criteria because of it?
- A teammate wants every command icon to follow the theme when the application switches themes. Which of the five IconCompare icons should become theme images instead, and which should not?

Ask me about anything that is ambiguous before you start.

Then review what came back

Code that compiles is not yet code that is right. The interesting questions are design questions: is this the Wisej.NET control the job actually calls for, does it follow the lifecycle the framework gives you, is the authoritative state on the server rather than parked in the browser because that was convenient? Read the result as a reviewer, not as a customer.

This module gives you the checklist. Answer each one against the code you just received:

- ✓ Referencing a pack of three thousand icons adds one assembly to the deployment. What does that cost the browser, and when?
- ✓ One of your two recoloured icons ignored the colour suffix. What is it about that artwork, and what would you change in the pack selection criteria because of it?
- ✓ A teammate wants every command icon to follow the theme when the application switches themes. Which of the five IconCompare icons should become theme images instead, and which should not?

Anything you had to correct is worth writing down. A short note of the conventions and decisions you settled on, pasted into the first message of your next session, is worth more than any single clever prompt.



No reference sample for this course yet — yours is the first

The [LearnWisej samples repository](#) carries worked builds for most courses, but not yet for this one. That means there is nothing to compare against and no shortcut: the specification and your own review are the whole safety net. Keep what you build — and if it turns out well, we would like to see it.

Three prompts to take it further

Point your assistant at the project you just built and try these. Each one is a review-and-extend exercise, not a rewrite.

- ✓ **Make it survive a bad day.** *"Review this IconDesk project for the failure modes it does not handle yet — bad or missing input, a dependency that is slow or unavailable, work arriving after the user navigated away, and two actions racing each other. For each one, tell me first whether it is a real risk in Wisej.NET 4 and why, and only then write the fix. Where you decide something needs no handling, say so and explain the reasoning rather than adding defensive code."*
- ✓ **Add the second case.** *"Extend this IconDesk project with a second case that follows the same pattern as the first — another entity, screen or control of the same shape, wired the same way. Factor out what is genuinely shared into a base class or service, but only where it earns its place; two similar things do not always want an abstraction. Show me the diff and justify each thing you extracted."*
- ✓ **Prove it actually works.** *"Go through this project deliverable by deliverable against the module specification at <https://learnwisej.com/assets/courses/icons-and-images/specs/module-04.md> and show me the evidence that each one is genuinely done — the code path, what to click, and what I*

should see. Where a deliverable is only partly met, say so plainly instead of arguing it through. Then add whatever is missing."



Carry it into the lab

The hands-on lab that closes this module asks for the same deliverables you just had the assistant produce. Use what you have here as the shape of your answer and the evidence behind it — not as the answer itself. The lab is graded on your reasoning, and that part has to be yours.