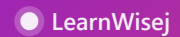


AI Coding Exercise

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.



The assistant is the fast pair of hands. You are still the architect.

An assistant can produce this module's sample in minutes. What it cannot do is decide whether the result is the right design for your application — that call stays with you. You supply the judgement: brief it properly, then check what comes back against what Module 2 just taught you.

What you are asking for

Build an ImageLab page in IconDesk that makes the two pipelines visible side by side. Show one raster asset four times with SizeMode set to Normal, Zoom, Cover and StretchImage so the cropping and distortion differences are obvious, add a PictureBox that pulls a remote JPEG with LoadAsync and keeps the page responsive while it arrives, and place one SVG assigned through ImageSource at 16, 24, 32 and 48 pixels to prove it stays sharp. Finish with a handler that reports, for every PictureBox on the page, whether its picture came from Image or from ImageSource.

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

- ✓ ImageLab page showing one raster asset in Normal, Zoom, Cover and StretchImage size modes
- ✓ PictureBox that fetches a remote JPEG with LoadAsync without blocking the page
- ✓ One SVG assigned through ImageSource and rendered at 16, 24, 32 and 48 pixels
- ✓ Report handler that names, for each PictureBox, whether Image or ImageSource supplied the picture
- ✓ Format note listing which lab assets travel through System.Drawing and which stay resource strings

Why it is shaped this way

"Which image formats does Wisej.NET support?" is the question every team asks in week one, and the honest answer depends on which API path the picture takes. Answer it with one vague list and you get a stretched logo on a high-DPI laptop, a page that freezes while a product photo downloads from a slow host, and a bug report because an.ico file will not resolve through a resource string. This module replaces the vague list with two precise ones, each attached to the control property that consumes it.

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
System.Drawing pipeline	The Image property holds a System.Drawing.Image in server memory; Wisej.NET sends it to the browser as PNG data. Its documented sources are bitmap, metafile, icon, JPEG, GIF and PNG files.
Image-source pipeline	A string the client resolves itself: a relative or absolute URL, a named theme image, or a resource.wx path. The documented resource.wx image extensions are .jpg, .jpeg, .png, .gif, .svg and .bmp.
PictureBox.Image	The entry point for pixels your C# code already owns or has just generated, such as a chart drawn with Graphics.
PictureBox.ImageSource	The entry point for a picture the browser can fetch itself. A short string travels instead of image bytes, which is why SVG belongs here.
PictureBox.SizeMode	The six values Normal, AutoSize, CenterImage, StretchImage, Zoom and Cover, which decide whether a picture is clipped, centred, distorted, fitted or cropped.
Load and LoadAsync	Load(url) updates the picture from a URL synchronously; LoadAsync(url) performs the same fetch asynchronously, so a slow host does not hold the screen.
Vector versus raster	An SVG is markup drawn at whatever size the layout asks for, and suitable monochrome artwork can be recoloured; a raster file is a fixed grid of pixels that only ever gets resampled.

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 2 is about. You can open [module-02.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 2 – "Bitmaps, SVGs and PictureBox: Formats, Sizing and Loading" – 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-02.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: Build an ImageLab page in IconDesk that makes the two pipelines visible side by side. Show one raster asset four times with `SizeMode` set to Normal, Zoom, Cover and `StretchImage` so the cropping and distortion differences are obvious, add a `PictureBox` that pulls a remote JPEG with `LoadAsync` and keeps the page responsive while it arrives, and place one SVG assigned through `ImageSource` at 16, 24, 32 and 48 pixels to prove it stays sharp. Finish with a handler that reports, for every `PictureBox` on the page, whether its picture came from `Image` or from `ImageSource`.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - ImageLab page showing one raster asset in Normal, Zoom, Cover and `StretchImage` size modes
 - `PictureBox` that fetches a remote JPEG with `LoadAsync` without blocking the page
 - One SVG assigned through `ImageSource` and rendered at 16, 24, 32 and 48 pixels
 - Report handler that names, for each `PictureBox`, whether `Image` or `ImageSource` supplied the picture
 - Format note listing which lab assets travel through `System.Drawing` and which stay resource strings
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:

- Your raster asset is wider than it is tall and the box is square. Which of Zoom and Cover loses part of the picture, which leaves empty space, and which one would you defend for a customer logo?
- The remote JPEG host is unreachable for ten seconds. What does the user of your page experience with `LoadAsync`, and what would they have experienced with the synchronous Load?
- Your report handler says a control's picture came from `ImageSource`, but the designer file also assigns `Image` on the same control. Which one is actually drawn, and how would you prove it rather than guess?

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:

- ✓ Your raster asset is wider than it is tall and the box is square. Which of Zoom and Cover loses part of the picture, which leaves empty space, and which one would you defend for a customer logo?
- ✓ The remote JPEG host is unreachable for ten seconds. What does the user of your page experience with LoadAsync, and what would they have experienced with the synchronous Load?
- ✓ Your report handler says a control's picture came from ImageSource, but the designer file also assigns Image on the same control. Which one is actually drawn, and how would you prove it rather than guess?

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.

- ✓ **Review it like a pull request.** *"Walk me through this IconDesk code as if you were reviewing a colleague's pull request and did not write it. Where does state live, what happens end to end on one user action, what would you send back for changes, and which parts would you not be comfortable putting in front of a customer? Be specific and quote the lines you mean — do not summarise the design back to me."*
- ✓ **Swap a piece without moving the seams.** *"Pick the part of this IconDesk project that is most obviously a stand-in for something real — a fake data source, an in-memory store, a simplified service — and replace it with a production-shaped implementation. The contract either side of it must not change: same types, same calls, same behaviour under failure. Tell me what the real implementation forced you to handle that the stand-in did not."*
- ✓ **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."



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.