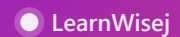


AI Coding Exercise

Build this module's SupportDesk 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 — including against our own reference build on GitHub.



You are not going to type this sample. You are going to direct it.

Almost nobody writes Wisej.NET code from a blank file any more — an assistant writes the first draft and the developer decides whether it is right. That only works if you know the material, which is what the rest of this module gave you. Here you hand that knowledge to ChatGPT or Claude, get this module's sample back as a working Visual Studio solution, and review it the way you would review a colleague's pull request.

What you are asking for

Optimise the Support Desk ticket browser: enable EF Core logging in development, record the SQL and timing of the unoptimised search, replace entity loading with projection, add AsNoTracking to grid queries, add paging with a page size of 50, replace lazy navigation access with projection or Include, and document the measured before-and-after results in your lab notes.

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

- ✓ EF Core logging enabled in development with the unoptimised SQL and timing recorded
- ✓ Grid query rewritten as a no-tracking projection
- ✓ Paging with a page size of 50 and indexed filters
- ✓ No lazy-loading N+1 behaviour left in display code
- ✓ Lab notes documenting the measured before-and-after improvement

Why it is shaped this way

The Support Desk ticket browser worked in Module 3 with fifty seed tickets. With fifty thousand it will not, unless the query does less work in the right place. The failure is quiet: a grid that takes three seconds because formatting code touched a navigation property per row, a session holding thousands of tracked entities nobody will edit, a search that scans a table. This module is about doing less work, in SQL, on purpose, and proving it with a log rather than a hunch.

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
projection	A Select that returns exactly the fields the screen needs, such as TicketListItem, so the database joins the related names in one statement.
Include and filtered Include	Eager loading of related entities with the root; a filtered Include narrows a collection (the last five comments) so an editor loads a controlled aggregate.
explicit loading	Loading a navigation later, on demand, through the entry's Collection or Reference and LoadAsync, when the user expands a detail section.
lazy loading and N+1	Navigation properties loaded transparently on first access; in a grid that is one hidden query per row or per navigation access.
AsNoTracking and identity resolution	Results that bypass the change tracker for read-only screens; AsNoTrackingWithIdentityResolution still returns one instance per entity when the same row appears repeatedly.
query-level paging	Where, OrderBy, Skip, Take and Select composed before ToListAsync, with a separate CountAsync for the total.
EF Core logging	The generated SQL of every command with its duration, through LogTo or Microsoft.Extensions.Logging.

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 6 is about. You can open [module-06.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 SupportDesk 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 6 – "Async Loads, Related Data, Filtering, and Performance" – of the LearnWisej Data Binding with EF Core 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/ef-core-binding/specs/module-06.md>

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

1. Project "SupportDesk", 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: Optimise the Support Desk ticket browser: enable EF Core logging in development, record the SQL and timing of the unoptimised search, replace entity loading with projection, add `AsNoTracking` to grid queries, add paging with a page size of 50, replace lazy navigation access with projection or `Include`, and document the measured before-and-after results in your lab notes.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - EF Core logging enabled in development with the unoptimised SQL and timing recorded
 - Grid query rewritten as a no-tracking projection
 - Paging with a page size of 50 and indexed filters
 - No lazy-loading N+1 behaviour left in display code
 - Lab notes documenting the measured before-and-after improvement
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 log showed 151 statements for one screen of 50 tickets. Which line of code caused the 150 extra ones, and why did nothing in the application warn you?
- The editor needs the ticket, its customer and category, and the last five comments; the comments panel opens the full history on demand. Which loading strategy serves each of those three needs, and why is projection the wrong answer for the editor?
- A teammate proposes compiled queries and context pooling for the ticket browser before looking at the log. What would you measure first, and what would have to be true of the numbers before either change is worth making?

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 log showed 151 statements for one screen of 50 tickets. Which line of code caused the 150 extra ones, and why did nothing in the application warn you?
- ✓ The editor needs the ticket, its customer and category, and the last five comments; the comments panel opens the full history on demand. Which loading strategy serves each of those three needs, and why is projection the wrong answer for the editor?
- ✓ A teammate proposes compiled queries and context pooling for the ticket browser before looking at the log. What would you measure first, and what would have to be true of the numbers before either change is worth making?

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.



We already built this one — compare yours with ours

The Module 6 reference implementation lives in the public samples repository. Build yours first, then open ours: the differences are the interesting part.



[Open the Module 6 sample on GitHub](#)

Or clone the whole set with `git clone https://github.com/iceteagroup/LearnWisej-Samples` and open `Data Binding with EF Core Course/Module 6`.



Our sample is a starting point, not a finished product

It was built the same way you are building yours — by prompt, then corrected — and it deliberately stops where the module stops. Expect stand-ins where production code would go, thin error handling and no tests. Treat that as an invitation: use what you learned in this module, plus the assistant, to push it further. Improving code you did not write is the skill this course is actually teaching.

Three prompts to take it further

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

- ✓ **Add the second case.** *"Extend this SupportDesk 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/ef-core-binding/specs/module-06.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."*
- ✓ **Review it like a pull request.** *"Walk me through this SupportDesk 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."*



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.