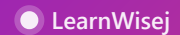


AI Coding Exercise

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.



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

Create the Customer, Agent, Category, Ticket and TicketComment entities for the Support Desk Data Console, configure relationships and delete behaviours with the Fluent API, configure RowVersion with IsRowVersion, add indexes for Status, DueDate, CustomerId and UpdatedAt, add a design-time factory, create the InitialCreate migration, and seed five customers, three agents, categories and at least fifty tickets into a local SQLite or SQL Server LocalDB database.

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

- ✓ Customer, Agent, Category, Ticket and TicketComment entities with relationships and delete behaviours
- ✓ RowVersion configured with IsRowVersion
- ✓ Indexes for Status, DueDate, CustomerId and UpdatedAt
- ✓ Design-time factory and the InitialCreate migration
- ✓ Seed data: five customers, three agents, categories and at least fifty tickets

Why it is shaped this way

The Support Desk Data Console will show tickets in a pageable grid, edit them in a modal form and let several agents work the same queue at once, and each screen is only as good as the model underneath it. A filter with no matching index becomes a table scan at fifty thousand tickets, two agents saving the same ticket silently lose one agent's work unless the row carries a concurrency token, and a schema patched by hand breaks the next deployment. The model is the contract between the UI and the database, so this module builds it explicitly and versions it as source.

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
binding-friendly entity	An entity with a predictable key, stable display fields, explicit navigation properties and small lookup tables, so a grid, a ComboBox and an editor can all bind to it.
TicketListItem view model	A flat projection of only the columns the grid shows, so the grid never receives the whole Ticket graph and stays independent from the editing unit of work.
Fluent API configuration	Mapping rules written in OnModelCreating (required strings, lengths, indexes, delete behaviour, precision, defaults) so persistence rules live next to the DbContext.
RowVersion concurrency token	A database-generated byte array marked IsRowVersion; EF Core adds its original value to the UPDATE's WHERE clause, so a stale save updates zero rows and throws DbUpdateConcurrencyException.
delete behaviour	The OnDelete rule of a relationship (Restrict, SetNull, Cascade) that decides whether deleting a customer is refused, unassigns tickets, or removes comments too.
migration	A C# class plus the model snapshot describing one reviewed schema change, committed to source control and applied in order to bring any database up to the model.
design-time factory	An IDesignTimeDbContextFactory that tells the dotnet ef tools how to construct the context without running the Wisej.NET application startup.

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 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 2 – "Modeling, DbContext Configuration, and Migrations" – 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-02.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: Create the Customer, Agent, Category, Ticket and TicketComment entities for the Support Desk Data Console, configure relationships and delete behaviours with the Fluent API, configure RowVersion with `IsRowVersion`, add indexes for Status, DueDate, CustomerId and UpdatedAt, add a design-time factory, create the InitialCreate migration, and seed five customers, three agents, categories and at least fifty tickets into a local SQLite or SQL Server LocalDB database.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Customer, Agent, Category, Ticket and TicketComment entities with relationships and delete behaviours
 - RowVersion configured with `IsRowVersion`
 - Indexes for Status, DueDate, CustomerId and UpdatedAt
 - Design-time factory and the InitialCreate migration
 - Seed data: five customers, three agents, categories and at least fifty tickets
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:

- You delete a customer who still has open tickets. What happens with the delete behaviour you configured, and what would have happened with Cascade?
- Two agents load ticket SD-1042, and both click Save a second apart. Which save throws, what does the UPDATE statement's WHERE clause contain, and what does the user see?
- Why does `IsRequired()` on Title not give the user an "Enter a title" message, and where will that message come from later in the course?

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:

- ✓ You delete a customer who still has open tickets. What happens with the delete behaviour you configured, and what would have happened with Cascade?
- ✓ Two agents load ticket SD-1042, and both click Save a second apart. Which save throws, what does the UPDATE statement's WHERE clause contain, and what does the user see?
- ✓ Why does IsRequired() on Title not give the user an "Enter a title" message, and where will that message come from later in the course?

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



[Open the Module 2 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 2`.



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.

- ✓ **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-02.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."*
- ✓ **Swap a piece without moving the seams.** *"Pick the part of this SupportDesk 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."*



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.