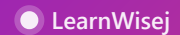


AI Coding Exercise

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 7 just taught you.

What you are asking for

Make the Support Desk Data Console safe for multiple users: add RowVersion to the edit model and its hidden state, set OriginalValue before saving an existing entity, reproduce a conflict by editing the same ticket in two browser sessions, catch DbUpdateConcurrencyException, build a conflict list of proposed and database values, add Reload and Overwrite paths according to policy, generate an idempotent SQL migration script with deployment notes and environment-specific connection strings, keep sensitive-data logging off outside development, then complete the capstone review.

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

- ✓ RowVersion carried in the edit model and set as OriginalValue before saving
- ✓ Reproducible concurrency conflict caught as DbUpdateConcurrencyException
- ✓ Conflict dialog listing proposed and database values with Reload and Overwrite paths
- ✓ Idempotent SQL migration script with deployment notes and environment-specific connection strings
- ✓ Sensitive-data logging disabled outside development and the capstone review completed

Why it is shaped this way

Two operators open ticket T-1042 in two browser sessions; one closes it, the other raises its priority, and whichever save lands second decides what the database remembers. Locking rows is not an option in a web application, so the Support Desk Data Console must let both users work, detect the collision at save time, explain it and let a person decide. This module also covers transactions, configuration, migrations, logging, tests and the capstone.

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
optimistic concurrency	Users edit freely; the application detects at save time that someone else changed the row since it was loaded, instead of locking rows.
RowVersion token	A database-generated concurrency token; EF Core puts its original value in the UPDATE's WHERE clause, so a changed row updates zero rows.
OriginalValue	The version the editor loaded, restored onto the tracked entity before SaveChangesAsync so EF Core compares against what the user saw.
DbUpdateConcurrencyException	Thrown when an expected row was not affected; its Entries hold the proposed values, and GetDatabaseValuesAsync fetches what the database holds now.
conflict dialog	A Wisej.NET form listing field, your value, database value and original value, offering Reload, Overwrite when policy permits, or Merge.
execution strategy	EF Core's retry for transient failures; an explicit transaction must run inside it so a retry can replay the whole unit of work.
idempotent migration script	SQL from dotnet ef migrations script --idempotent that checks the migrations history before each step, so it is safe on any environment.

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 7 is about. You can open [module-07.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 7 – "Concurrency, Transactions, Deployment, Diagnostics, and Capstone" – 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-07.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: Make the Support Desk Data Console safe for multiple users: add `RowVersion` to the edit model and its hidden state, set `OriginalValue` before saving an existing entity, reproduce a conflict by editing the same ticket in two browser sessions, catch `DbUpdateConcurrencyException`, build a conflict list of proposed and database values, add `Reload` and `Overwrite` paths according to policy, generate an idempotent SQL migration script with deployment notes and environment-specific connection strings, keep sensitive-data logging off outside development, then complete the capstone review.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `RowVersion` carried in the edit model and set as `OriginalValue` before saving
 - Reproducible concurrency conflict caught as `DbUpdateConcurrencyException`
 - Conflict dialog listing proposed and database values with `Reload` and `Overwrite` paths
 - Idempotent SQL migration script with deployment notes and environment-specific connection strings
 - Sensitive-data logging disabled outside development and the capstone review completed
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:

- If you left out the `OriginalValue` line, what would the two-session test show, and why would it look like success while silently losing the first operator's change?
- Which resolution paths would you offer an ordinary agent and which a supervisor, and what must happen to the grid and the `RowVersion` in the edit model after each path?
- Three production instances start after a deployment. In your design, which of them applies the migration and when, and what would go wrong if `Migrate()` ran in each instance's startup?

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:

- ✓ If you left out the OriginalValue line, what would the two-session test show, and why would it look like success while silently losing the first operator's change?
- ✓ Which resolution paths would you offer an ordinary agent and which a supervisor, and what must happen to the grid and the RowVersion in the edit model after each path?
- ✓ Three production instances start after a deployment. In your design, which of them applies the migration and when, and what would go wrong if Migrate() ran in each instance's startup?

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



[Open the Module 7 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 7`.



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-07.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.