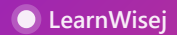


AI Coding Exercise

Build this module's FieldOps 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.



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

Create the FieldOps solution with a work-order list, a work-order detail page and a status update flow that work in a plain browser. Write the delivery decision record: compare browser, PWA, desktop and mobile shells for the technicians' needs, choose the targets and their order, and define the capability checks the app will use to enable device features.

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

- ✓ FieldOps solution with list, detail and status update
- ✓ Delivery model comparison table
- ✓ Delivery decision record with target order
- ✓ Capability check design
- ✓ Architecture diagram with one server and multiple shells

Why it is shaped this way

FieldOps is a field-service application: technicians receive work orders, capture photos and signatures, navigate to sites and work with patchy connectivity. Built as an ordinary Wisej.NET application it already runs in any browser, which is enough for the dispatchers in the office and not enough for a technician standing in a plant room with one bar of signal and a phone in a glove. The question this module answers is not "can we build it" but "how does it reach the people who use it" — and that answer is an architecture decision, not a packaging afterthought.

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
delivery model	The route the application takes to a user: a browser tab, an installed PWA, a desktop window or a store-distributed mobile app. It changes distribution, updates and device access — never the C# that builds the screen.
host browser engine	Every model renders the same server-driven UI in a browser engine: the user's own browser, or an embedded one — WebView2 on Windows, WebKit on macOS and iOS, the system WebView on Android.
Wisej.NET Hybrid	The framework's .NET MAUI-based shell that hosts your existing Wisej.NET application on Android, iOS and desktop and adds a bridge to device features such as camera, geolocation, files and notifications.
device bridge	The two-way channel between your server-side C# and the native shell: the server asks for a device action, the shell performs it and the result comes back to your code as an event, not as a return value.
capability check	A runtime test of what the current shell can actually do, used to show, hide or substitute a feature so one screen behaves correctly in every delivery model.
application shell cache	The static HTML, CSS, scripts and icons a PWA service worker can keep locally. It is what still loads without a network — the interactive, server-rendered part of the app is not in it.
update asymmetry	The server can be redeployed at any moment and every session gets the new build on reload, while a desktop installer or a store package needs a release and an install first.

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 1 is about. You can open [module-01.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 FieldOps 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 1 – "Hosting Models Beyond the Browser" – of the LearnWisej Hybrid & Native Packaging 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/hybrid-native-packaging/specs/module-01.md>

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

1. Project "FieldOps", 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 FieldOps solution with a work-order list, a work-order detail page and a status update flow that work in a plain browser. Write the delivery decision record: compare browser, PWA, desktop and mobile shells for the technicians' needs, choose the targets and their order, and define the capability checks the app will use to enable device features.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - FieldOps solution with list, detail and status update
 - Delivery model comparison table
 - Delivery decision record with target order
 - Capability check design
 - Architecture diagram with one server and multiple shells
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:

- A technician loses signal halfway through updating a work order. Describe exactly which parts of FieldOps can still be shown from a cached shell, which cannot, and what your screen should do at the moment the connection drops.
- Your decision record puts the Android shell before the PWA. What would have to be true about the technicians' devices and network for that order to be wrong?
- The detail page needs a photo. Explain why a capability check plus an Upload fallback is a better design than a build-time switch that produces a mobile-only version of the screen.

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:

- ✓ A technician loses signal halfway through updating a work order. Describe exactly which parts of FieldOps can still be shown from a cached shell, which cannot, and what your screen should do at the moment the connection drops.
- ✓ Your decision record puts the Android shell before the PWA. What would have to be true about the technicians' devices and network for that order to be wrong?
- ✓ The detail page needs a photo. Explain why a capability check plus an Upload fallback is a better design than a build-time switch that produces a mobile-only version of the screen.

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.

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



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.