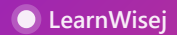


AI Coding Exercise

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.



Writing the code is the easy half now.

Getting working code for "Mobile Shells with .NET MAUI" out of a modern assistant takes one good prompt. Judging it, correcting it and extending it takes everything this module covered. That is the split this exercise is built around — you supply the specification and the review, the assistant supplies the typing.

What you are asking for

Build the FieldOps mobile shell with .NET MAUI: a hybrid page hosting the app on Android and iOS, a session that survives backgrounding and resumes on return, the Android back button mapped to in-app navigation, safe-area padding for notched devices, and a debug build running on one emulator per platform with screenshots of the work-order list.

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

- ✓ .NET MAUI hybrid shell project for Android and iOS
- ✓ Lifecycle handling for background and resume
- ✓ Back button and safe-area mapping
- ✓ Emulator builds for both platforms
- ✓ Screenshots from both emulators

Why it is shaped this way

FieldOps technicians work from a phone in a plant room, a basement or a van, and a browser tab is a poor container for that: no home-screen icon to hand out, no store or enterprise channel, and no route to the camera and GPS that Module 5 will need. The mobile operating system is also a far more aggressive host — it suspends the app the moment a call arrives, redraws it under a notch, and hands the technician a system back gesture FieldOps never asked for.

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
.NET MAUI hybrid shell	A small native app for Android and iOS whose single page hosts your Wisej.NET application in a web view, so the C# screens you already ship are what the technician sees.
native web view host	The platform browser engine embedded in that page — Android System WebView, WKWebView on iOS — running the same client a desktop browser gets, which is why no screen is rewritten for mobile.
hybrid bridge	The two-way channel between the hosted page and native code, so server-side C# can ask the device for something and receive the answer as an event rather than a page navigation.
MAUI application lifecycle	The events the OS raises around the app — the window created, activated, deactivated, stopped and resumed, plus OnSleep and OnResume — none of which a browser tab raises.
session survival window	The stretch of background time during which the Wisej.NET session is still alive on the server, so a resume can reconnect to existing state instead of starting the app over.
safe area insets	The per-device padding that keeps content clear of the notch, the status bar and the home-indicator strip, reported by the platform rather than guessed from the screen size.
back navigation mapping	The rule that decides what Android's back gesture does: move back inside the app's own navigation, or confirm leaving — never silently destroy a half-filled work order.

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 4 is about. You can open [module-04.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 4 – "Mobile Shells with .NET MAUI" – 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-04.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: Build the FieldOps mobile shell with .NET MAUI: a hybrid page hosting the app on Android and iOS, a session that survives backgrounding and resumes on return, the Android back button mapped to in-app navigation, safe-area padding for notched devices, and a debug build running on one emulator per platform with screenshots of the work-order list.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - .NET MAUI hybrid shell project for Android and iOS
 - Lifecycle handling for background and resume
 - Back button and safe-area mapping
 - Emulator builds for both platforms
 - Screenshots from both emulators
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 is halfway through a status update when a call comes in for three minutes. What must be true on the server and in the shell for the half-finished update to still be there afterwards, and what changes if the call lasts an hour?
- The shell knows nothing about which FieldOps screen is open. How does a back gesture decide between popping the detail page, closing a dialog and leaving the app, and where does that state actually live?
- Which of this module's changes would a technician still notice if they opened FieldOps in the phone's browser instead of the shell, and which would vanish? What does that tell you about where to put the safe-area padding?

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 is halfway through a status update when a call comes in for three minutes. What must be true on the server and in the shell for the half-finished update to still be there afterwards, and what changes if the call lasts an hour?
- ✓ The shell knows nothing about which FieldOps screen is open. How does a back gesture decide between popping the detail page, closing a dialog and leaving the app, and where does that state actually live?
- ✓ Which of this module's changes would a technician still notice if they opened FieldOps in the phone's browser instead of the shell, and which would vanish? What does that tell you about where to put the safe-area padding?

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.

- ✓ **Prove it actually works.** *"Go through this project deliverable by deliverable against the module specification at <https://learnwisej.com/assets/courses/hybrid-native-packaging/specs/module-04.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 FieldOps 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 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."



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.