

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.

● LearnWisej



Same module, built the way you will actually build things.

Every other section of this module explains how "Device APIs through the Hybrid Bridge" works. This one asks you to produce it — by directing an assistant with the module's own specification, then verifying the result runs and does what the lab brief describes. It is the workflow you will use on real work, practised on something whose right answer you already know.

What you are asking for

Add device features to FieldOps: a Take Photo action that uses the device camera in the native shell and falls back to an Upload control in the browser, a signature capture uploaded as an image, a Navigate action that reads the device location and opens the maps app, a push notification when a work order is assigned, and a capability check that shows or hides each action per shell.

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

- ✓ Camera capture with browser upload fallback
- ✓ Signature capture and upload
- ✓ Geolocation and maps hand-off
- ✓ Push notification on assignment
- ✓ Capability check matrix across shells

Why it is shaped this way

A FieldOps technician photographs a corroded valve, collects a signature on the glass and drives to the next address — while the dispatcher who assigned the job watches the same work order in a desktop browser. Every control in FieldOps is a C# object on your server, which cannot touch the camera, the GPS chip or the notification tray. This module is the round trip that closes that gap, and what the screen does where the round trip does not exist.

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
device bridge	The message channel a hybrid shell exposes to the page it hosts: your server asks the shell to perform a device operation, and the shell answers with a message the app turns back into a server-side event.
host object	The object a WebView injects into the hosted page so JavaScript there can call native code. Server-side C# reaches it only through the app's JavaScript interop.
capability descriptor	The features a shell reports when a session starts — camera, location, notifications, biometrics — cached and consulted before any device action is shown.
progressive enhancement	Designing each screen so the plain-browser path is a complete feature and the native path is a better version of it, never the only one that works.
just-in-time permission	Asking the platform for camera, location or notification access at the moment the user presses the button that needs it, right after a sentence explaining why.
bridge result event	The asynchronous reply — photo bytes, coordinates or a refusal — arriving on a later request than the click that asked for it, so it needs its own push.
storage outside the web root	Writing captured photos and signatures where no URL maps to them, then serving them through a handler that checks whether this session may see that 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 5 is about. You can open [module-05.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 5 – "Device APIs through the Hybrid Bridge" – 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-05.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: Add device features to FieldOps: a Take Photo action that uses the device camera in the native shell and falls back to an Upload control in the browser, a signature capture uploaded as an image, a Navigate action that reads the device location and opens the maps app, a push notification when a work order is assigned, and a capability check that shows or hides each action per shell.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Camera capture with browser upload fallback
 - Signature capture and upload
 - Geolocation and maps hand-off
 - Push notification on assignment
 - Capability check matrix across 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:

- The camera result arrives forty seconds after the click that asked for it. Explain which request it arrives on, what session state it must find in order to know what to do with the bytes, and why the picture box does not appear in the browser without an explicit push.
- A technician refused the location permission permanently three weeks ago on their own phone. Describe exactly what pressing Navigate should do today, and name the one thing the app must never do in response.
- Take Photo works in the mobile shell and the Upload fallback works in desktop Chrome. Name one thing that is genuinely different for the user between those two paths, and argue whether that difference is acceptable for a field technician.

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:

- ✓ The camera result arrives forty seconds after the click that asked for it. Explain which request it arrives on, what session state it must find in order to know what to do with the bytes, and why the picture box does not appear in the browser without an explicit push.
- ✓ A technician refused the location permission permanently three weeks ago on their own phone. Describe exactly what pressing Navigate should do today, and name the one thing the app must never do in response.
- ✓ Take Photo works in the mobile shell and the Upload fallback works in desktop Chrome. Name one thing that is genuinely different for the user between those two paths, and argue whether that difference is acceptable for a field technician.

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.

- ✓ **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."*
- ✓ **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."



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.