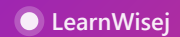


AI Coding Exercise

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

Complete the FieldOps capstone: sign the Windows installer and the Android and iOS packages, prepare an enterprise distribution for the mobile shells and a store listing draft, define a compatibility rule between server versions and shell versions with a minimum-version check on startup, add crash reporting and a usage event for each device feature, and deliver a release checklist covering all four delivery models.

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

- ✓ Signed Windows, Android and iOS packages
- ✓ Enterprise distribution setup and store listing draft
- ✓ Server and shell version compatibility check
- ✓ Crash and usage telemetry
- ✓ Release checklist for all delivery models

Why it is shaped this way

FieldOps finishes this course as one server and four ways to reach it: a browser tab, an installed PWA, a Windows desktop shell and a mobile shell on Android and iOS. The three packaged ones are files an operating system refuses to install unless an identity it trusts signed them, and each travels a different channel with its own review queue and rollout speed. You can redeploy the server before lunch; the iOS build a technician is holding may be three weeks behind it, and it still has to work.

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
Authenticode signature	The signature a code-signing certificate applies to a Windows installer or MSIX, proving the file was not altered after signing and naming a vetted publisher.
countersignature timestamp	A trusted timestamp recorded at signing time, so the installer keeps validating after the signing certificate itself expires.
Android keystore and upload key	The key that signs an Android build; with Play App Signing you sign with an upload key and Google re-signs with the app signing key it holds for that package.
Apple distribution certificate and provisioning profile	The certificate identifies your team; the profile binds an App ID, that certificate, the entitlements and, outside the App Store, the allowed devices.
enterprise distribution	Delivering a private build to one organisation's devices — managed Google Play, Apple Business Manager custom apps, an MSIX pushed by Intune — instead of publishing publicly.
minimum shell version	The oldest shell build, per platform, the current server still supports; anything below it is stopped on connect before it reaches any data.
device-feature usage event	A record written whenever a shell capability runs — camera, signature, location, push, offline flush — carrying the outcome plus the shell platform and version.

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 6 is about. You can open [module-06.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 6 – "Signing, Distribution, Updates and the FieldOps Capstone" – 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-06.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: Complete the FieldOps capstone: sign the Windows installer and the Android and iOS packages, prepare an enterprise distribution for the mobile shells and a store listing draft, define a compatibility rule between server versions and shell versions with a minimum-version check on startup, add crash reporting and a usage event for each device feature, and deliver a release checklist covering all four delivery models.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Signed Windows, Android and iOS packages
 - Enterprise distribution setup and store listing draft
 - Server and shell version compatibility check
 - Crash and usage telemetry
 - Release checklist for all delivery models
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's Android shell is six weeks old and the server has shipped four times since. Which of those four server changes should have raised the minimum shell version, and how would you have known at the time?
- The Windows installer was signed but not timestamped, and the certificate expired last month. What do existing installations do, what do new downloads do, and what is the shortest path back to a trusted installer?
- The camera usage event shows a sharp rise in the browser fallback path on iOS only. Name three explanations that fit the data, and say which extra field in the event would tell them apart.

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's Android shell is six weeks old and the server has shipped four times since. Which of those four server changes should have raised the minimum shell version, and how would you have known at the time?
- ✓ The Windows installer was signed but not timestamped, and the certificate expired last month. What do existing installations do, what do new downloads do, and what is the shortest path back to a trusted installer?
- ✓ The camera usage event shows a sharp rise in the browser fallback path on iOS only. Name three explanations that fit the data, and say which extra field in the event would tell them apart.

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.