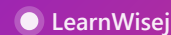


AI Coding Exercise

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.



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 2 just taught you.

What you are asking for

Turn FieldOps into a PWA: add the manifest with icons, theme color and standalone display, register a service worker that caches the shell and static assets, show an install prompt after the first successful session, display an offline page with the last synced work-order summary when the server is unreachable, and implement the update flow that reloads when a new version is available.

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

- ✓ Web app manifest with icons and colors
- ✓ Service worker caching the shell
- ✓ Install prompt after first session
- ✓ Offline page with last synced summary
- ✓ Update flow and version check

Why it is shaped this way

FieldOps technicians do not open a browser and type a URL. They tap an icon on a phone that has been in a van all morning, in a plant room where the signal drops to nothing halfway through a job. A progressive web app gives them that icon, a window without browser chrome and a page that still says something useful when the network disappears — but FieldOps is a Wisej.NET app, so its user interface lives on the server, and pretending otherwise is how a PWA turns into a support ticket.

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
web app manifest	A small JSON file, linked from the start-up page, that tells the browser the app's name, icons, colors, display mode and start URL — without it the browser has nothing to install.
display mode	The display member of the manifest. standalone launches FieldOps in its own window with no address bar, which is what makes an installed app stop feeling like a bookmark.
maskable icons	Icons declared with purpose: "maskable" and safe padding, so Android can crop them into its own shape without clipping the artwork.
service worker	A script the browser runs beside the page, served from the app's own origin over HTTPS, that can answer the page's requests from a cache when the connection is slow or gone.
application shell	The static files a session needs before the server can say anything: the start-up page, the client scripts, the theme and the icons. It is what the worker is allowed to cache.
cache-first vs network-only	The routing rule inside the worker's fetch handler: static shell files come from the cache first, while everything the Wisej.NET session exchanges with the server must go straight to the network, never a cached copy.
update lifecycle	A new worker installs while the old one still controls the page and then waits. skipWaiting, clients.claim and a controlled reload are what turn that waiting worker into the version the technician actually runs.

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 2 is about. You can open [module-02.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 2 – "Progressive Web App Setup" – 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-02.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: Turn FieldOps into a PWA: add the manifest with icons, theme color and standalone display, register a service worker that caches the shell and static assets, show an install prompt after the first successful session, display an offline page with the last synced work-order summary when the server is unreachable, and implement the update flow that reloads when a new version is available.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Web app manifest with icons and colors
 - Service worker caching the shell
 - Install prompt after first session
 - Offline page with last synced summary
 - Update flow and version check
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 reviewer asks why the service worker does not simply cache every response so FieldOps keeps working underground. Explain what a replayed Wisej.NET session response would actually do to the technician's screen, and where the honest boundary lies.
- Your install offer works on your desk but never appears on the technician's phone. Name three separate conditions that could be responsible, and say how you would tell them apart from the device.
- A technician has the app installed and open when you deploy a new build. Describe every step between your deployment and that phone running the new shell, and identify which step would silently fail if the cache name never changed.

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 reviewer asks why the service worker does not simply cache every response so FieldOps keeps working underground. Explain what a replayed Wisej.NET session response would actually do to the technician's screen, and where the honest boundary lies.
- ✓ Your install offer works on your desk but never appears on the technician's phone. Name three separate conditions that could be responsible, and say how you would tell them apart from the device.
- ✓ A technician has the app installed and open when you deploy a new build. Describe every step between your deployment and that phone running the new shell, and identify which step would silently fail if the cache name never changed.

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.

- ✓ **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."*
- ✓ **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-02.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."



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.