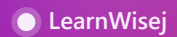


Mobile Shells with .NET MAUI

Lesson Guide · Lab / Exam Guide



Lesson Guide

Hosting the app in a .NET MAUI hybrid shell for Android and iOS, navigation and back-button behaviour, app lifecycle and platform quirks. What this means for a team shipping a Wisej.NET app beyond the browser tab — and how to read this module.



Production focus

Hosting the app in a .NET MAUI hybrid shell for Android and iOS, navigation and back-button behaviour, app lifecycle and platform quirks.

1. The production problem

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.

2. Key concepts

Concept	Why it matters
.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 <code>OnSleep</code> and <code>OnResume</code> — 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.

3. Lesson narrative

One shell project, two stores, the same server

The FieldOps mobile shell is a .NET MAUI application with one job: create a window, host a web view, point it at the FieldOps server. Wisej.NET Hybrid support ships that shell as a project you build and adjust rather than write from scratch, and the page it hosts is the ordinary FieldOps application from Module 1. Nothing about `Wisej.Web`, the Visual Studio designer or `Application.Session` changes; the controls still live on the server and the web view still renders their mirror image.

One MAUI project targets Android and iOS at once, and the platform folders under it hold only what genuinely differs: the Android activity and manifest, the iOS `Info.plist` and entitlements, the icons and splash screens. Keep the server address in one configuration value the shell reads at startup, so debug points at a laptop, test at staging and release at production without a code change.

That split is also the economics of the shell. FieldOps ships new server code the moment you deploy it; the shell only needs rebuilding when the shell itself changes — a new icon, a new permission, a new bridge capability. Anything that can live on the server should, because the server is the part you can fix this afternoon while a store review queue is measured in days.

Backgrounding is the normal case, not the failure case

On a phone the app is interrupted constantly: a call, a part number checked in another app, a screen locked while climbing a ladder. Each raises the MAUI lifecycle sequence — the window deactivated, then

stopped, then resumed on the way back, with the application-level `OnSleep` and `OnResume` overrides firing alongside. Handle them explicitly, because the default behaviour of a web view evicted from memory is to reload the page, and a reload is a new Wisej.NET session: the half-typed meter reading is gone.

The design that works treats a short background as a pause and a long one as a restart. On stop, record the time and let the session idle. On resume, compare the elapsed time against the session timeout configured for the FieldOps server; if the session should still be alive, reconnect and let the server push the current state back rather than navigating the web view again. If the gap is longer than the timeout, or the platform reports the app was terminated for memory, restore deliberately: reload, then reopen the work order using an identifier the shell persisted locally. Treat a low-memory warning exactly like a stop.

```
protected override Window CreateWindow(IActivationState state)
{
    var window = base.CreateWindow(state);
    window.Stopped += (s, e) => ShellState.Backgrounded(DateTime.UtcNow);
    window.Resumed += (s, e) => ShellState.Resume(); // reconnect, don't reload
    return window;
}

protected override void OnSleep() => ShellState.Backgrounded(DateTime.UtcNow);
protected override void OnResume() => ShellState.Resume();
```

The server side of the contract matters just as much: keep the FieldOps session timeout long enough to cover a realistic interruption, and make the work-order editor save a draft on every field change rather than only on submit.

Back, notches and the strip at the bottom

Android's back gesture is a system-level event, and ignoring it means the platform default: the activity closes, so a technician who swipes back from a work-order detail page loses the app instead of returning to the list. Intercept back navigation in the shell and route it to the web app first — the detail page pops back to the list, an open dialog closes — falling through to a confirmation prompt only when nothing is left to pop. The state that answers "is there anything to go back to" lives on the server, so the shell must ask the app rather than guess: keep a small, explicit navigation stack in FieldOps itself. iOS has no back button, so the app's own back affordance must exist regardless.

Safe areas are the other visible platform quirk. A notched phone reserves the top strip and a modern iPhone reserves a home-indicator strip at the bottom; content drawn there is hidden or untappable. Read the insets the platform reports and apply them as padding around the hosted content, so the FieldOps app bar sits below the status bar and the status-update buttons sit above the home indicator. Pad the host rather than writing fixed pixel values into the Wisej.NET layout, and the same screens stay correct in a plain browser, on a device with no notch, and in landscape.

Emulators, devices and two build commands

The build loop is per-platform and worth setting up before writing any shell code. Android needs the SDK and at least one Android Emulator image; a debug run is `dotnet build -t:Run -f net8.0-android` or the equivalent target in Visual Studio. iOS needs a Mac in the loop — the iOS Simulator and a build such

as `dotnet build -f net8.0-ios` — even when you develop on Windows and pair to that Mac remotely. Hot restart shortens the cycle, and because most FieldOps changes are server-side, many need no rebuild at all: change the C#, restart the server, pull down in the shell.

Debugging spans two runtimes: breakpoints in the shell's C# cover lifecycle and bridge calls, breakpoints in the server code cover the screens. When the emulator cannot reach your development server, the cause is almost always the loopback address — an emulator's `localhost` is the emulated device, not your machine — or a self-signed certificate the platform refuses. Then prove the shell on real hardware: touch targets, the keyboard covering a field, and eviction under memory pressure all behave differently there, and the lab's screenshots from both emulators are the baseline you compare that device run against.



Packaging habit

The app is still a Wisej.NET web app served by your server; the shell is a delivery vehicle. Keep every feature working in a plain browser, then add device capabilities as progressive enhancements behind a capability check.



Learning objectives

By the end of this module you'll be able to:

- ✓ Host the Wisej.NET app in a .NET MAUI hybrid shell targeting Android and iOS.
- ✓ Handle mobile lifecycle events — background, resume, low memory — without losing the session.
- ✓ Map platform navigation, the back button and safe areas to the web app.
- ✓ Set up emulators, device debugging and the build pipeline for both platforms.

Lab / Exam Guide

What you'll build in the hands-on lab, the suggested approach, and the required deliverables.



Hands-on lab

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.

Starting point

You open the FieldOps solution from Module 1 — the work-order list, the work-order detail page and the status update flow, already running in a plain browser and, since Module 2, installable as a PWA. Nothing in those screens changes in this lab; you add a second head next to the web project, a .NET MAUI shell that hosts the same server-rendered app on a phone.

Suggested approach

- ✓ Add the MAUI shell project to the FieldOps solution, target Android and iOS, and read the server address from configuration so debug and release builds point at different hosts.
- ✓ Host the FieldOps app in the shell's web view and confirm the work-order list, the detail page and a status update all work end to end before touching anything platform-specific.
- ✓ Handle the window stopped, resumed and low-memory events: record the time the app went to the background and, on resume inside the session timeout, reconnect instead of reloading the page.
- ✓ Intercept Android back navigation and route it into the FieldOps navigation stack, prompting for confirmation only when there is nothing left to go back to; apply the reported safe-area insets as padding around the hosted content.
- ✓ Run a debug build on the Android Emulator and on the iOS Simulator, capture the work-order list on each, and note every difference you had to account for per platform.

Required deliverables

- ✓ .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

Student review questions

- ✓ 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?



Packaging habit

The app is still a Wisej.NET web app served by your server; the shell is a delivery vehicle. Keep every feature working in a plain browser, then add device capabilities as progressive enhancements behind a capability check.