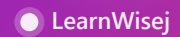


Hybrid, PWA, Offline, and Device-Aware Wisej.NET Applications

Lesson Guide · Lab / Exam Guide



Lesson Guide

Connected/offline modes, local data, device APIs, sync conflict patterns, field scenarios. What this means for an advanced Wisej.NET team — and how to read this module.



Production focus

Connected/offline modes, local data, device APIs, sync conflict patterns, field scenarios.

1. The production problem

Some enterprise workflows require mobile, tablet, field, kiosk, or offline operation. Advanced Wisej.NET developers must decide when Hybrid/PWA patterns fit and how to preserve a single application model without ignoring device realities.

2. Key concepts

Concept	Why it matters
Wisej.NET Hybrid	The framework's support for running the same application in the browser, as a desktop or mobile app, and with native device access.
connected mode	The standard model where the app runs on the server and the device is a client. It is simplest, but it requires a network connection.
offline/local app	Running the application locally on the device so it keeps working without a connection, at the cost of managing local data and synchronization.
device abstraction	Placing camera, file system, GPS, and other device features behind service interfaces so screens do not depend on a specific platform.
local SQLite	A lightweight embedded database on the device that stores working data while offline and feeds the synchronization queue.
sync queue	A durable list of local changes waiting to be sent to the server when connectivity returns, processed in order and safely retried.
conflict resolution	The rules for what happens when the same record changed both locally and on the server while offline, decided by the product, not by chance.
PWA shell	The progressive web app manifest and service worker that let the application install on a device and load its shell without a network.

3. Lesson narrative

Offline is a product decision

Offline support is not a checkbox. It changes identity, authorization freshness, data storage, conflict handling, audit timing, support procedures, and deployment. Advanced developers limit offline scope to workflows that genuinely need it.

Keep device code behind services

Camera, scanner, clipboard, notifications, and network status should be exposed through services. The rest of the application should call business-friendly interfaces instead of scattering device-specific code across screens.

Four application shapes

Wisej.NET Hybrid lets one application model run in four shapes. In connected mode the application runs on a server and the browser, or a native shell around a browser, is a thin client; the device needs a network connection for every action. In remote mode a native shell on the device connects to that same server application but gains access to device features through the shell. In local mode the application itself is packaged with the device, running its server process on the device, so it works without a network. Offline mode is local mode with local data and a synchronization boundary to the central system.

Each step along that line trades simplicity for independence. Connected mode has one copy of the data and one place to enforce rules. Offline mode has a copy on every device, and the rules must hold on each of them. The right shape is the least independent one that satisfies the workflow. A dispatcher at a desk stays connected. A technician in a basement with no signal needs an offline queue for the work they complete there.

State the shape in the architecture note before designing screens, because it decides where identity, data, and permissions live.

Offline workflows and synchronization boundaries

An offline-capable workflow is designed around what the device is allowed to hold and what it is allowed to change. The Field Technician mode in the lab caches the technician's assigned work orders in a local SQLite store, and only those. Completing a work order offline does not update the record; it appends an `OfflineCommand` to a local queue with an explicit `SyncState` of pending. The screen shows the pending state so the technician knows the completion has not reached the office yet.

The synchronization boundary is the point where the queue is replayed against the server. Each command is sent in order, executed by the same server service that handles the online case, and marked synced or conflicted. Because the server service is the same, every rule, permission check, and audit entry applies to offline work exactly as it does to online work. The device never talks to the database directly.

Limit the scope aggressively. Every additional entity cached offline is more data on a device that can be lost, more sync cases to test, and more conflicts to resolve.

Device capabilities behind service interfaces

Camera, barcode scanner, GPS, local notifications, and network status reach the application through the Hybrid shell, and each behaves differently on a phone, a tablet, a kiosk, and a desktop browser. The lab hides all of them behind an `IDeviceServices` abstraction. A screen calls a method such as capture photo or read barcode, and the implementation for the current shape decides whether that means the native camera, a file picker, or a message that the feature is unavailable.

The abstraction pays off three times. Screens stay identical across shapes, so one form serves the office and the field. Tests can substitute a fake device service that returns a known photo or barcode. And a new device or shell only requires a new implementation, not a search through every form for platform-specific calls.

Treat device output as untrusted input. A scanned barcode is validated on the server like any other value, and a GPS position is a hint for the user, never proof of where the work was done.

Conflicts, stale permissions, and offline security

A conflict occurs when the same record changed on the device and on the server while they were apart. In the lab's failure path the dispatcher cancels a work order while the technician, offline, completes it. Neither version is simply wrong, so the sync conflict screen shows both, records both, and asks a person to decide. The resolution rules, such as which roles may override and whether a cancelled order can still be completed, are product decisions documented before the code is written.

Permissions go stale offline. The technician's role may have been removed while they were disconnected, and the device still holds yesterday's permission set. The sync boundary handles this by re-evaluating every queued command against current permissions on the server, so a revoked user's queued commands are rejected and audited rather than applied. The device also refreshes its cached permissions on every reconnect.

- Local data is encrypted at rest and wiped on logout or after a lock-out period.
- Cached credentials expire, and a stolen device cannot replay commands after that.
- Every offline command is audited on the server with its original device timestamp and its sync time.



Advanced production habit

Do not stop when the screen looks correct. Explain state ownership, security implications, failure paths, and production behavior before considering the module complete.



Learning objectives

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

- ✓ Explain connected, remote, local, and offline Hybrid application shapes.
- ✓ Design offline-capable workflows with local storage and synchronization boundaries.
- ✓ Use device capabilities through explicit service abstractions.
- ✓ Handle sync conflicts, stale permissions, and offline security tradeoffs.

Lab / Exam Guide

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



Hands-on lab

Design and prototype a Field Technician mode for EnterpriseOps. Add device-aware layout, offline work-order cache, local completion queue, reconnect sync simulation, and conflict display.

Suggested approach

- ✓ Read the production problem and identify the architecture risk before touching the UI.
- ✓ Create or update the relevant service, command, model, or component boundary first.
- ✓ Build the Wisej.NET screen or component using the established project standards.
- ✓ Add a failure-path demonstration, not only the happy path.
- ✓ Record what changed in the learning log and update the relevant checklist.

Required deliverables

- ✓ Hybrid/offline architecture note
- ✓ Device service abstraction
- ✓ Offline queue model
- ✓ Sync conflict screen
- ✓ Field-mode usability checklist

Student review questions

- ✓ What data is allowed offline?
- ✓ How are permissions refreshed?
- ✓ What happens when local changes conflict with server changes?



Advanced production habit

Do not stop when the screen looks correct. Explain state ownership, security implications, failure paths, and production behavior before considering the module complete.