

Observability, Diagnostics, Performance, and Session Profiling

Lesson Guide · Lab / Exam Guide

 LearnWisej

Lesson Guide

Health, logs, metrics, traces, client/server timing, memory/session growth. What this means for an advanced Wisej.NET team — and how to read this module.



Production focus

Health, logs, metrics, traces, client/server timing, memory/session growth.

1. The production problem

Advanced teams must diagnose production problems without remote guessing. They need safe diagnostics, structured logs, performance measurements, and session/memory review practices.

2. Key concepts

Concept	Why it matters
structured logging	Writing log entries as named fields rather than free text, so they can be searched, filtered, and correlated by tools in production.
correlation ID	A unique identifier carried through every log entry for one request or job, so a single user action can be followed across services and servers.
diagnostics page	An operator-only screen that exposes session counts, memory, active jobs, and recent errors, giving insight without attaching a debugger.
health check	A lightweight endpoint that reports whether the application and its dependencies are working, used by load balancers and monitoring.
performance budget	Agreed limits for response time, payload size, and memory per session, so regressions are caught against a number rather than a feeling.
session memory	The server memory each live session consumes. It must stay small and predictable, because a Wisej.NET server hosts many sessions at once.
disposal review	Checking that timers, event handlers, DbContexts, and background tasks are released when a form or session ends, so memory does not leak.
safe error handling	Catching and logging exceptions with full detail on the server while showing users a generic message, so internals are never exposed.

3. Lesson narrative

Production has no debugger

A production runbook should answer what version is running, what environment it is in, which services are healthy, what errors occurred, and which correlation ID connects the user report to logs.

Diagnostics must be safe

A diagnostics screen should be role-protected and redact secrets. It can show environment, version, feature flags, active theme, service status, health check result, queue status, and recent safe logs.

Diagnostics pages that help without exposing secrets

A diagnostics page is the first place a support engineer looks when a ticket arrives. It should answer, in one screen, which version and build is running, which environment and node served the session, which feature flags are on, whether dependent services are healthy, and how long recent operations took. The lab's page shows version, node, flags, and performance budgets side by side so that the answer to most first questions is a glance.

The page is also an attack surface, so its contents are chosen by type rather than by hand. The `DiagnosticSnapshot` in the lab only carries fields that are safe to display: no connection strings, no API keys, no personal data, no full stack traces. Values that could be sensitive, such as a storage account name,

are redacted before they reach the view model. The page is behind a role check, and opening it is itself audited.

A good review question is whether the page could be shown on a projector in front of a customer. If any value would have to be covered, it should not be there.

Structured logs, correlation, and safe messages

Structured logging means every entry is a set of named fields rather than a sentence. An entry for a slow query records the operation name, the tenant, the user, the duration in milliseconds, the page size, and the correlation ID as separate values. The lab's `OperationTimer` wraps each service operation and writes such an entry when it completes, so the log can later be filtered to every operation over two seconds in one tenant.

The correlation ID is what ties the pieces together. It is created when a user action starts, carried in the command context through every service call, passed to background jobs, and written into audit entries and error reports. When a user reports that an approval failed at 14:32, support finds the correlation ID from the error they were shown and follows it through every log line that action produced.

The message the user sees is a different thing from the log entry. It says that the operation could not be completed, gives the correlation ID, and suggests what to do next. It never shows the exception text, the SQL, or the server path, because those help an attacker more than they help the user.

Measuring what users feel

Performance is measured at the points where users wait. Startup is the time from the first request to a usable main page. Screen load is the time from a menu click to a rendered form. A data query is the time the service spends fetching a page of results. Binding refresh is the time between the data arriving and the grid showing it. Background job timing covers queue wait and execution, and update timing covers how long a pushed change takes to appear in the browser.

Each of those gets a budget, which is a number the team agrees to keep under, recorded in the lab's performance budget table. The budget turns a vague complaint into a measurable regression. The lab's failure path shows the value of this: a query that took 2,340 ms against a 500 ms budget is red on the table, and its structured log entry reveals a page size of 5,000 rows. Fixing the page size turns the row green, and the log proves it.

Measure in production as well as in development, because data volume, network distance, and concurrent sessions change the numbers. A timing panel that only works on a developer machine is a demo, not a diagnostic.

Session memory and disposal mistakes

Every live session holds memory on the server for as long as it exists, and a Wisej.NET server hosts many sessions at once. A form that keeps a 50,000-row list in a field, a cached report, or a large file buffer multiplies that cost by the number of users. The session memory review in the lab lists what each session retains, why, and for how long, and challenges every item over a few megabytes.

Disposal mistakes are the second source of growth. A timer that is started in a form and never stopped keeps the form alive after it closes. An event handler attached to a shared service holds a reference to a

disposed control. A DbContext kept in a field survives the operation it was created for. A background task that captures a form in a closure keeps everything the form references. Each of these leaks a little, and over a working day the little adds up.

- Which fields on this form hold collections, and are they cleared on close.
- Which timers, subscriptions, and tasks does this form start, and where are they stopped.
- Which objects are retained for the whole session, and does each one need to be.



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:

- ✓ Design diagnostics pages that help operations without exposing secrets.
- ✓ Add structured logging, correlation IDs, and user-safe error messages.
- ✓ Measure startup, screen load, data query, binding refresh, background job, and update timing.
- ✓ Identify memory/session growth risks and disposal mistakes.

Lab / Exam Guide

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



Hands-on lab

Add observability to EnterpriseOps: structured logging, correlation IDs, diagnostics page, health status, performance timing panel, and a session-memory review checklist. Simulate a slow query and show how it is diagnosed.

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

- ✓ Diagnostics page
- ✓ Structured log example
- ✓ Correlation ID propagation
- ✓ Performance budget table
- ✓ Memory/session audit notes

Student review questions

- ✓ Can support identify the exact version and tenant?
- ✓ Does diagnostics expose secrets?
- ✓ Which objects are retained for the whole session?



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.