

Security Architecture: Identity, SSO, Authorization, Audit, and Secure Deployment

Lesson Guide · Lab / Exam Guide

● LearnWisej

Lesson Guide

Identity integration, permission model, audit trails, safe HTML, secure configuration. What this means for an advanced Wisej.NET team — and how to read this module.



Production focus

Identity integration, permission model, audit trails, safe HTML, secure configuration.

1. The production problem

Server-side architecture reduces many browser attack surfaces, but advanced systems still need identity integration, service-level authorization, audit trails, safe rendering, and secure operational configuration.

2. Key concepts

Concept	Why it matters
OIDC/SSO boundary	The point where an external identity provider hands the application a verified identity. Everything after that boundary is the application's responsibility.
claims mapping	Translating provider-specific claims into the application's own roles and permissions, so authorization code never depends on one identity provider's format.
permission service	A single service that answers whether the current user may perform an action, so the rule lives in one place instead of every screen.
service-level authorization	Enforcing permissions inside application services, not only in the UI, so a hidden button is never the only thing protecting an operation.
audit log	A tamper-evident record of who did what and when for important actions, so security events can be investigated and compliance can be proven.
AllowHtml review	Checking every place raw HTML is allowed in a control to make sure user-supplied content is sanitized, because unreviewed HTML is a cross-site scripting risk.
secret handling	Keeping connection strings, keys, and tokens in secure configuration or a vault, never in source control or client-visible settings.
secure headers	HTTP response headers such as content security policy and strict transport security that reduce the browser's attack surface for the application.

3. Lesson narrative

Authentication is not authorization

A logged-in user is not automatically allowed to approve a work order, view another tenant, export data, or run an admin task. The permission check belongs in the service that executes the action.

Session context bridges identity and UI

After authentication, the application should map identity into a session context: user ID, display name, tenant, roles, permissions, culture, theme, and correlation IDs. This context helps screens adapt while services still enforce rules.

An identity and authorization model for the enterprise

Enterprise authorization has more dimensions than a role name. The same person may be a dispatcher in one tenant, a read-only auditor in another, and allowed to approve work orders only below a cost limit. A model that can express that starts with three layers: identity, which says who the user is; membership, which says which tenants and roles they hold; and permissions, which are the fine-grained actions a role grants, such as `ApproveWorkOrder` or `ExportData`.

The lab turns this into a permission matrix: rows are permissions, columns are roles, and each cell is granted, denied, or conditional. The matrix is the artifact that product owners, security reviewers, and

developers can all read, so it should exist before the code does. Workflow steps map onto it too. Submitting an escalation, approving it, and closing it may each require a different permission, and the matrix makes that explicit.

Keep the model tenant-first. A permission is always evaluated inside a tenant context, so a user who holds a role in tenant A gets nothing in tenant B. The `PermissionService.Demand` call in the lab checks the tenant guard first and only then consults the role store.

Authentication gates, session context, and services

An external identity provider, reached through an OpenID Connect or SSO login, tells the application who the user is. That is where the provider's job ends. The application then maps the incoming claims into its own session context: user ID, display name, tenant, roles, permissions, and a correlation ID. Because the mapping runs once at login and is stored in the session, screens can read it cheaply and services can rely on it without repeating the claims parsing.

The gate itself must sit in front of every entry point, not only the main page. Background jobs, WebMethod calls, postback handlers, and diagnostics pages all need to know who is calling. A service should never read identity from a UI control, a query string, or a hidden field, because all of those can be edited by the user. It reads the session context, and the session context was built from verified claims.

Design the context so that it can be refreshed. Roles change while sessions are open, and a long-running session must be able to reload permissions without forcing a new login. A review question for every service method is simple: which line proves the caller is allowed to be here.

Audit trails for sensitive commands

An audit trail answers who did what, to which record, in which tenant, when, and with what result. It is written by the service that executes the command, inside the same operation, so an action cannot succeed without being recorded. The lab's audit log screen shows every `Demand`, granted or denied, and every sensitive command such as an approval, an export, or a permission change.

Denials are as important as successes. A wrongly enabled button that lets a user request `ExportData` is caught by the service, which refuses with a named error and writes a denied entry. Repeated denials from one account are a signal that someone is probing the application, and they are only visible if denials are logged. The entry should carry the correlation ID so it links to the structured logs from Module 11.

- Actor, tenant, and impersonation flag if any.
- Command name, target record, and the before and after values that matter.
- Outcome: granted, denied, or failed, with the named reason.
- Timestamp and correlation ID, written in the same transaction as the change.

A security checklist for the whole surface

Server-side rendering removes many browser-side risks, but a Wisej.NET application still ships HTML, accepts uploads, sets cookies, writes logs, reads secrets, and sits behind a proxy. Each of those is a place where a small mistake becomes an incident. The lab's hardening checklist is the document that turns those places into review items that are checked on every release rather than remembered occasionally.

Start with HTML. Any control that renders raw HTML from user input must sanitize it, and the `AllowHtml` review in the lab lists every such control. Uploads need size limits, type checks, and storage outside the web root. Cookies should be marked secure and HTTP-only, with a same-site policy. A content security policy limits which scripts may run. Logs must never contain passwords, tokens, or full personal records. Secrets live in a vault or protected configuration, never in source control. A reverse proxy must forward the original scheme and client address so the application sees real values.

Treat the checklist as living code: add an item every time a review finds a gap, and record who checked each item for each release.



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 an identity and authorization model for enterprise roles, tenants, and workflows.
- ✓ Integrate authentication gates with session context and application services.
- ✓ Build audit trails for sensitive commands.
- ✓ Create security checklists for HTML, uploads, cookies, CSP, logs, secrets, and reverse proxies.

Lab / Exam Guide

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



Hands-on lab

Add enterprise authentication simulation, claims-to-permission mapping, tenant-role enforcement, service-level authorization, safe HTML review, export approval, and audit logging for sensitive actions.

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

- ✓ Identity mapping design
- ✓ Permission matrix
- ✓ Permission service implementation
- ✓ Audit log screen
- ✓ Security hardening checklist

Student review questions

- ✓ Where is authorization enforced?
- ✓ Which UI elements render HTML?
- ✓ Can an enabled button bypass service rules?



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.