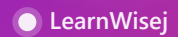


Wisej.NET 4 Project Modernization and Migration Strategy

Lesson Guide · Lab / Exam Guide



Lesson Guide

Upgrade planning, compatibility analysis, incremental migration, regression safety. What this means for an advanced Wisej.NET team — and how to read this module.



Production focus

Upgrade planning, compatibility analysis, incremental migration, regression safety.

1. The production problem

Many advanced Wisej.NET teams maintain large existing applications. They must modernize project structure, target frameworks, resources, themes, and deployment assumptions without gambling with business continuity.

2. Key concepts

Concept	Why it matters
migration inventory	A complete list of projects, packages, themes, resources, and custom components in the existing app, so nothing is discovered halfway through the upgrade.
compatibility matrix	A table of target frameworks, Wisej.NET versions, and third-party packages showing which combinations are supported before any code changes.
incremental upgrade	Moving one project or layer at a time with a working build after every step, instead of a single big-bang conversion that cannot be verified.
resource mapping	Tracking where images, scripts, themes, and embedded resources live before and after the migration so nothing breaks silently in production.
theme migration	Porting custom themes and mixins to the current theme format and verifying every screen renders correctly under the new engine.
regression harness	A repeatable set of automated and scripted checks that proves existing behavior still works after each migration step.

3. Lesson narrative

Modernization is risk management

An upgrade should produce a better application, but it should not become an uncontrolled rewrite. Advanced developers treat migration as a sequence of verifiable changes: compile, run, compare behavior, check themes, verify sessions, review deployment, and measure performance.

Inventory before action

Before touching code, list target framework, package versions, designer dependencies, custom widgets, themes, embedded resources, startup files, authentication entry points, deployment targets, and third-party integrations. This inventory becomes the migration map and the final review checklist.



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:

- ✓ Build a migration inventory for existing Wisej.NET 3.x or legacy-style projects.
- ✓ Identify project, resource, theme, deployment, and API changes that require explicit testing.
- ✓ Design an incremental migration path with fallback points.
- ✓ Create a regression checklist that protects behavior, not just compilation.

Lab / Exam Guide

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



Hands-on lab

Create a migration dossier for the Intermediate TicketOps Console and extend it into the Advanced project baseline. Include current-state inventory, target-state inventory, risk matrix, screen regression plan, and rollback strategy.

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

- ✓ Migration inventory table
- ✓ Compatibility/risk matrix
- ✓ Regression test plan for ten key flows
- ✓ Rollback plan
- ✓ Modernization decision memo

Student review questions

- ✓ What exact behavior must remain unchanged?
- ✓ Which migration step can be rolled back independently?
- ✓ Which screens are most likely to hide timing or state assumptions?



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.