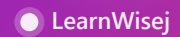


Advanced Workflow UX: Wizards, Modal Orchestration, and Compensation

Lesson Guide · Lab / Exam Guide



Lesson Guide

Complex business flows, typed results, undo/compensation, resumable workflows. What this means for an advanced Wisej.NET team — and how to read this module.



Production focus

Complex business flows, typed results, undo/compensation, resumable workflows.

1. The production problem

Enterprise workflows rarely fit into a single Save button. Advanced screens coordinate approvals, assignments, validations, uploads, and external checks while preserving user confidence.

2. Key concepts

Concept	Why it matters
workflow orchestration	A service that owns the sequence, rules, and state of a multi-step business process, so the flow is not buried across several forms.
wizard state	The accumulated answers and validation status of a multi-step wizard, held in one typed object rather than scattered across controls.
typed result	A return value that explicitly models success, validation failure, and error outcomes, so callers must handle each path instead of relying on exceptions.
compensation command	An operation that undoes the effects of an earlier step when a later step fails, keeping the system consistent without a single giant transaction.
resumable workflow	A process that persists its progress so a user can close the browser and continue later, or a failure can be recovered from the last good step.
failure path matrix	A table listing every step, what can go wrong there, and how the workflow responds, so failure handling is designed rather than improvised.

3. Lesson narrative

Workflow is not a sequence of forms

A wizard is only the visible part of a business process. The advanced design question is where the workflow state lives, which service owns transitions, and how the UI resumes after interruption.

Typed results keep code honest

Dialogs and wizard pages should return typed intent, not magic flags scattered across controls. This makes workflow transitions testable and easier to review.

Typed commands in, typed results out

The Escalation Wizard collects a reason, attachments, an approver, a due date, and notification options across six pages. None of those pages runs business logic. They fill one `EscalationCommand`, and when the user confirms, a workflow service executes it and returns a `WorkflowResult`. The command is a plain object with every field typed and validated, so it can be built in a unit test as easily as in the wizard.

The result is typed as well. It says whether the escalation was created, which validations failed and on which field, whether an external step such as notification succeeded, and what the user should do next. Callers switch on those outcomes instead of catching exceptions or inspecting strings. A wizard page that receives a validation failure can jump to the right step and highlight the field; a batch caller can log the same result without any UI.

The lab's first review question is whether the workflow can be tested without the wizard. With a typed command and result, the answer is yes, and that test exists before the wizard does.

Wizards and modals that stay thin

A modal dialog or a wizard page is a form, and forms in this course are thin. The page collects values, performs the input validation needed to enable the next button, and writes the values into the wizard state object. It does not decide whether the approver is allowed, whether the due date is acceptable for this priority, or whether an attachment is required. Those are business rules, and they live in the workflow service where they apply to every caller.

The wizard state object is the visible part of that discipline. It is a single typed object holding everything collected so far and which steps are complete. Pages read from and write to it, the final page turns it into the command, and because it is one object it can be persisted, which is what makes the workflow resumable after a browser refresh. The lab's second question, what state survives refresh, is answered by whatever is in that object and stored on the server.

A modal that calls a repository, or a wizard page that holds a `DbContext`, has hidden business logic in a Form and will be duplicated the first time the same workflow is needed without the wizard.

Compensation where transactions cannot reach

Creating an escalation persists a record, sends a notification through an external system, and writes an audit entry. A database transaction covers the first and the last, but not the notification, which has already been sent when the audit write fails. Distributed transactions across external systems are rarely available and rarely worth their cost, so the workflow uses compensation instead: for every step that cannot be rolled back, define a compensating action that undoes or neutralizes it.

The lab's failure path is the notification failing after the persist succeeded. The escalation is kept, because it is valid, and a compensation command records that the notification is outstanding and schedules a retry or a manual follow-up. The reverse case, audit failing after notification, is compensated by recording the audit gap explicitly and alerting, because a sent notification cannot be unsent.

Compensation is designed per step and written down before the code. The question for each step is what the system should look like if this step fails after the previous ones succeeded.

A matrix for every failure path

The failure path matrix is a table with one row per step of the workflow and one column per way it can end: confirm, cancel, timeout, validation failure, and external failure. Each cell says what the system does, what the user sees, what is persisted, and what is audited. Filling the matrix exposes the paths nobody had designed, such as what happens when the approver selection times out or when the user cancels after attachments were uploaded.

The matrix is also the test plan. Every cell becomes at least one test of the workflow service, and the cells that involve the UI become manual checks in the demo script. When a reviewer asks what happens if the external check is down, the answer is a cell reference, not a guess.

- Confirm: the command executes, the result is shown, the audit entry is written.
- Cancel: collected state is discarded or kept as a draft, and uploads are cleaned up.
- Timeout: the workflow can be resumed from its persisted state, and nothing is half-applied.
- Validation failure: the user is returned to the failing step with the field highlighted.
- External failure: the compensation for that step runs, and the outcome is recorded.



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 multi-step workflows with typed commands and results.
- ✓ Use modal and wizard flows without hiding business logic in Forms.
- ✓ Implement compensation or undo where true transactions are not possible.
- ✓ Document confirm, cancel, timeout, validation failure, and external failure paths.

Lab / Exam Guide

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



Hands-on lab

Create an Escalation Wizard that collects reason, attachments, approver, due date, and notification options. Implement orchestration in a service, not in the wizard pages, and add compensation for a simulated notification failure.

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

- ✓ Wizard screen flow
- ✓ Workflow service
- ✓ Typed command/result objects
- ✓ Failure path matrix
- ✓ Compensation example

Student review questions

- ✓ Can the workflow be tested without the wizard?
- ✓ What state survives browser refresh?
- ✓ What happens after notification succeeds but audit logging fails?



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.