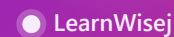


High-Volume Data UX, Server Filtering, and Batch Operations

Lesson Guide · Lab / Exam Guide



Lesson Guide

Large grids, paging, filtering, sorting, batch edits, user feedback. What this means for an advanced Wisej.NET team — and how to read this module.



Production focus

Large grids, paging, filtering, sorting, batch edits, user feedback.

1. The production problem

Enterprise users expect fast grids and reliable bulk actions. Loading everything into memory or rebinding blindly works in demos but fails under realistic data volume.

2. Key concepts

Concept	Why it matters
server-side paging	The database returns only the requested page of rows, so a grid can browse millions of records without loading them into memory or the browser.
search projection	A lean read model containing only the columns a search screen displays, instead of full entities with every navigation property.
grid state	The current page, sort, filter, and selection of a grid, kept on the server so it survives refreshes and can be restored predictably.
virtualized thinking	Designing every list and grid as if the data set is unbounded, so rows are fetched, rendered, and released on demand.
batch command	One server operation that applies a change to many selected records at once, with validation and transaction handling done in one place.
partial failure	When some items in a batch succeed and others fail. The operation must report exactly which ones failed and why, rather than an all-or-nothing error.
responsive feedback	Progress indicators, disabled controls, and status messages that keep users informed during long operations instead of leaving the screen frozen.

3. Lesson narrative

Large data is a workflow problem

The goal is not to show one million rows. The goal is to help the user find the right record, understand its status, and act safely. Search, filters, saved views, summaries, and batch actions matter more than raw row count.

Project for the grid

A grid row often needs display text, status icons, computed age, permission flags, and counts. A projection model keeps the grid fast and clear without contaminating domain entities with presentation details.

Filtering and paging on the server

The first screen of the Enterprise Work Queue loads exactly one page of rows, and that fact defines the design. The grid does not receive a list; it receives a `PagedResult` containing the rows for the current page and the total count. Filters, sort order, and page number are sent to the server as a `WorkQueueQuery`, and the query service translates them into a database query that returns only those rows. The database does the filtering, because it has the indexes and the memory for it.

Saved views make server-side filtering usable. A dispatcher who checks overdue high-priority orders every morning saves that filter once, and the saved view is a stored query definition, not a cached result. Sort persistence and the current page are part of the grid state kept on the server, so a refresh restores the same view instead of starting over.

The review question is the lab's first one: how many records are loaded for the first screen. If the answer is more than one page, the design has not started yet.

Projections instead of entities

A grid row needs display values: a work order number, a customer name, a status label and icon, an age in days, and whether the current user may reassign it. The domain entity has none of those directly; it has foreign keys, timestamps, and a status enumeration. Binding the entity to the grid forces the screen to compute the display values row by row, and forces the entity to grow presentation properties it should not have.

A search projection is a class shaped for the screen, such as `WorkQueueRow`, filled by the query service with a select statement that computes the display values in the database. It carries only the columns the grid shows, so a page of two hundred rows is a few kilobytes instead of a graph of entities. It also carries the permission flags the grid needs to enable or disable actions, computed once on the server.

Projections and entities change for different reasons. A new column on the grid changes the projection; a new business rule changes the entity. Keeping them apart means neither change breaks the other.

Batch operations that never fail silently

Reassigning fifty work orders is one user action and fifty business operations. The batch command in the lab carries the selected keys and the target technician, and the handler processes them one at a time, validating each, applying the change, and recording the result per row. A row can fail because the order was closed meanwhile, because the technician lacks a certification, or because another user changed it first. None of those should stop the other forty-nine.

Partial failure is therefore the expected outcome, not an error. The handler returns a report listing each row with succeeded, failed, or skipped and a reason. The screen shows progress while the batch runs and the report when it ends, and offers a retry for the failed rows only. The failure path in the lab is exactly this: reassign three, one fails, and the user sees which one and why.

- Is each row validated and applied independently, in its own transaction or a documented group.
- Does the result name every row and its outcome.
- Can the user retry the failures without repeating the successes.

Measuring perceived performance

Users judge a screen by how long they wait, not by how many rows it could hold. Perceived performance is measured at the moments they notice: the time from opening the screen to the first rows appearing, the time from changing a filter to the grid updating, the time from clicking a batch action to the first progress indication. The lab's performance notes record those numbers for the Work Queue with realistic data volumes and a realistic network.

Tuning follows the measurements. A slow first load usually means too many columns or a missing index for the default sort. A slow filter change usually means the filter is not translated to the database. A batch that feels frozen needs progress within the first second, even if the work itself takes a minute. Responsive feedback, a disabled button, a progress bar, and a status line cost almost nothing and change how long the wait feels.

Virtualized thinking is the habit behind all of this: design as if the data set has no upper bound, and every list, dropdown, and export will still work when it grows tenfold.



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 high-volume list screens with server-side filtering and paging.
- ✓ Create projections that separate grid display from domain models.
- ✓ Implement safe batch operations with progress, validation, and partial failure handling.
- ✓ Measure and tune perceived performance.

Lab / Exam Guide

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



Hands-on lab

Create an Enterprise Work Queue screen with server-side filters, saved views, paging, sort persistence, and a batch reassignment action. Add progress and a per-row result report for partial failures.

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

- ✓ Paged query service
- ✓ Search projection model
- ✓ Saved view definition
- ✓ Batch reassignment workflow
- ✓ Performance notes

Student review questions

- ✓ How many records are loaded for the first screen?
- ✓ What happens when one batch row fails?
- ✓ Can the user repeat the search after refresh?



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.