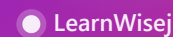


Complex Data-Bound Widgets: General Patterns with Kendo and DevExtreme Examples

Lesson Guide · Lab / Exam Guide



Lesson Guide

Integrate rich data widgets without making the course depend on one vendor library. What this means for a Wisej.NET integration developer — and how to read this module.



Production focus

Integrate rich data widgets without making the course depend on one vendor library.

1. The integration problem

Advanced libraries such as Kendo UI and DevExtreme provide powerful grids, pivot grids, spreadsheets, and data stores, but each has its own API, lifecycle, and data operation model.

2. Key concepts

Concept	How to use it
Data operations	Map paging, sorting, filtering, grouping, and editing to server handlers.
Vendor data source	Treat Kendo/DevExtreme as examples of a general integration pattern.

3. Production lesson

Kendo UI and DevExtreme are advanced examples because they contain rich widgets with real data behavior. They are not the definition of Wisej.NET integration. The same patterns apply to any serious JavaScript widget library: load the resources in the correct order, create the widget in the right container, pass options, wire events, feed data, handle remote operations, and expose a .NET API appropriate for the target application. For grids and pivot grids, the most important design decision is the data contract. A read-only pivot can return a JSON array or an OLAP-style response. An editable grid needs load, insert, update, and delete operations. Remote paging and filtering require request models. Layout serialization requires a separate state object. Exporting requires either client-side vendor functionality or a server-side pipeline. Kendo and DevExtreme have different APIs, but the Wisej.NET adapter responsibilities are the same.

Designing the data contract first

A rich grid or pivot widget is defined by the operations it performs on data, so the contract is designed before any vendor code is written. A read-only view needs a load operation that returns rows. A paged

view needs skip and take, or a page index and size. A sorted or filtered view needs a description of the sort and filter the widget applied. A grouped view returns aggregates, and a pivot returns a structure organized by row and column fields. An editable view adds insert, update, and delete, each with a key and the changed values.

The lab captures this as a typed `GridOperationRequest` on the server, holding skip, take, sort, and filter, and a result with the rows and the total count. Every vendor library, whatever it calls these things, can be mapped onto that request. Designing the request in .NET first keeps the server in control of what a filter may contain and how large a page may be.

Layout state, such as column widths and visible columns, is a separate contract, stored per user, never mixed into the data operations.

Mapping vendor operations to endpoints

Kendo's DataSource and DevExtreme's CustomStore differ in syntax, but both call a URL or a function for load, and both call again for insert, update, and delete when remote operations are enabled. The wrapper's job is to translate. For a URL-driven source, the InitScript points the vendor at the postback URL and the `WebRequest` handler reads the vendor's argument names, converts them to the `GridOperationRequest`, and returns the result in the shape the vendor expects.

For a function-driven source, the InitScript's functions call a `[WebMethod]` with typed arguments and return the awaited result to the vendor. Either way, the server side is the same handler with the same validation, and switching vendors changes only the translation in the InitScript.

Enabling remote operations is the moment the grid becomes a real integration. Until then, the vendor sorts and pages in memory and the server is only asked once; after that, every user interaction is a server call that must be fast and safe.

JSON options versus typed properties

Both Kendo and DevExtreme are configured through large option objects, and the fastest way to get a grid on screen is to assign such an object to Options directly from C#. That is right for the read-only pivot prototype in the lab, where the goal is to prove the widget runs and the data contract works.

A reusable grid control cannot stay that way. An untyped options bag has no validation, no defaults, no help text, and no way to know which of its hundred keys the application actually uses. Typed properties for the settings the application depends on, such as page size, editable, and column definitions, give the Designer something to show and reviewers something to check. The remaining vendor options can stay behind a protected extension point for the rare case that needs them.

The trade-off is speed now against maintainability later. Prototype with JSON, then promote every option that survives into a typed property before the control ships to other developers.

Two vendors, one pattern

Kendo and DevExtreme appear in this course because they are widely used and because their differences make the shared pattern visible. One uses a DataSource with transport settings, the other a CustomStore with load and modify functions. One names paging skip and take, the other names it similarly but shapes

the filter expression differently. Once both are mapped onto the same `GridOperationRequest`, the differences collapse into a few lines of translation in each InitScript.

That is the lesson to carry to any other library. The adapter responsibilities never change: load the resources in order, create the widget in the container, pass the options, wire the few events that matter, feed the data through a contract the server controls, handle remote operations, and expose a .NET API sized for the application. A team that can do this for two libraries can do it for a third without a new course.

Keep vendor-specific knowledge in the InitScript and the event translation, and keep the server contract vendor-free, so a licensing or roadmap change never forces a rewrite of application code.



Production habit

Keep Wisej.NET state ownership and client widget behavior clear. Document the client/server contract before relying on the integration in production.



Common mistake to avoid

Copying demo JavaScript into screens without documenting lifecycle, resources, event contracts, cleanup, or security boundaries.



Learning goals

By the end of this module you'll be able to:

- ✓ Design data contracts for read-only, editable, paged, filtered, sorted, grouped, and pivoted data.
- ✓ Map vendor data-source operations to Wisej.NET postback handlers or WebMethods.
- ✓ Understand why JSON options are fast but typed .NET properties are safer for reusable controls.
- ✓ Use Kendo and DevExtreme as reference implementations of the same general integration patterns.

Lab / Exam Guide

What you'll build in the hands-on lab, the recommended workflow, the required deliverables, and the self-check before you submit.



Hands-on lab

Build two advanced examples: a read-only pivot-style widget and an editable grid-style widget. One may use Kendo-style DataSource semantics, the other DevExtreme-style CustomStore semantics.

Recommended workflow

- ✓ Open the existing Wisej.NET integration course project and run it locally.
- ✓ Identify the server component, client widget, resource file, or endpoint involved in this module.
- ✓ Make one small change at a time and test it in the browser before moving on.
- ✓ Keep vendor-specific JavaScript inside the adapter or wrapper instead of scattering it across pages.
- ✓ Update the UI so the learner can clearly see success, failure, progress, or communication changes.
- ✓ Take screenshots or notes that prove the integration works and explain the design decision.

Required deliverables

- ✓ Read-only pivot integration plan
- ✓ Editable grid operation contract
- ✓ Load/update/insert/delete server handlers or stubs
- ✓ Event payload contracts for cell click and row update

Self-check before submitting

- ✓ Why are grids harder than gauges?
- ✓ What changes when remote operations are enabled?
- ✓ Why should Kendo and DevExtreme be taught as examples, not as the whole course?