CASE STUDY

Global Filtering Pattern

One filtering pattern for an enterprise product — so every module narrows and builds filters the same way, with a right-sized component for every data size in between.

01 — PROJECT OVERVIEW

Project overview

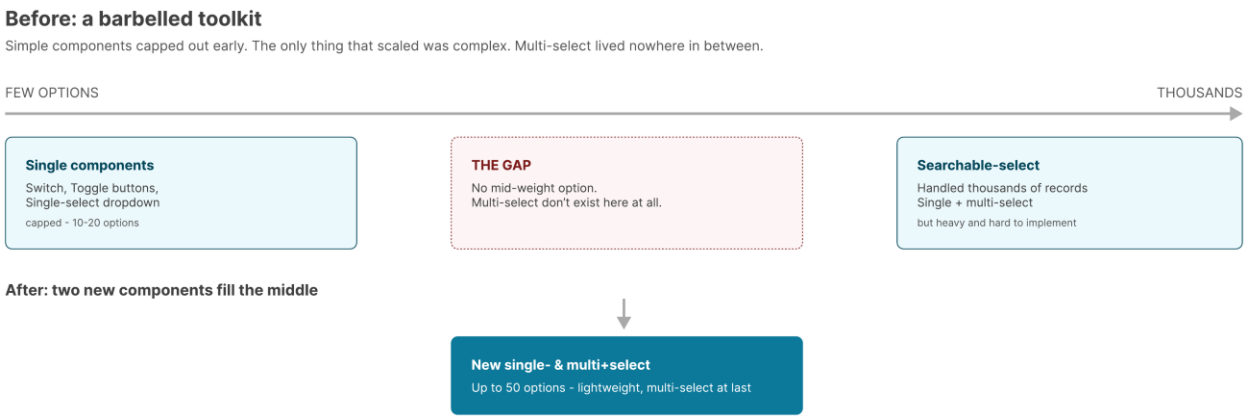
Problem statement

Filtering was fragmented on two axes at once. **Across modules**, each one had built its own filtering — or none — so the same action behaved differently in incident, assets, change, and the knowledge base, and nothing a user learned in one module carried to the next. **Inside the design system**, the toolkit was barbelled: a binary switch, quick toggle buttons, and single-select dropdowns at the simple end, and one large, complex searchable-select at the other — built because the simple components choked on the thousands of records our databases routinely returned. The middle was missing, and most painfully, **multi-select existed nowhere but inside that complex component**. Any module needing to filter by several values was forced into the heaviest, hardest-to-implement option in the system, no matter how little data it had.

ROLE Lead UX Designer, Design System
CONTEXT ITSM web app — B2B, 7,000+ organisations
SCOPE A system-wide pattern + components, consumed across modules
PARTNERS Asset Management designer, mobile / taskboard designer, engineering, accessibility
STATUS Shipped in core modules, rolling out

Solution

A single filtering pattern with two locations over one shared model: **standard filters** across the top of a dataset for the common, high-frequency choices, and an **advanced filter panel** at the side for depth — multiple conditions, ranges, and user-added criteria. To make it real, I filled the component gap with two new selects — single and multi — that scale to 50 options, so multi-select finally existed without reaching for the complex component, plus a guideline that tells any designer exactly which control to use for which filter at which data size.



02 — THE HARD PARTS

The challenges, and how I approached them

CHALLENGE

A hole in the middle of the toolkit. Simple dropdowns capped out around 10–20 options and multi-select only lived inside the complex searchable-select. Modules either over-reached for the heavy component or built their own.

APPROACH

I built two new components — a single-select and a multi-select — that handle up to 50 options cleanly, sitting between the trivial dropdowns and the searchable-select. Multi-select finally existed as a first-class, lightweight control. Above 50, you still escalate to the searchable-select in the advanced panel; below it, there's now a right-sized tool for every case.

CHALLENGE

Standard and advanced filtering blurring into each other. Once both a top bar and an advanced panel exist, the obvious failure is the same filter showing up in both — and users wondering which one is real.

APPROACH

One rule, strictly held: **each filter belongs to exactly one location.** High-frequency and simple lives in the top bar; low-frequency, complex, or detailed lives in the advanced panel; nothing appears twice. That single constraint is what keeps the pattern legible no matter how much filtering a module piles on.

CHALLENGE

Designing for one live use case and a product full of future ones. Real demand for deep, user-built filtering came from exactly one place — Asset Management. But building only for them would have produced another module-specific solution, the very thing I was trying to end.

APPROACH

I treated Asset Management as the proving ground, not the spec. I designed the advanced panel against their real needs while pulling in a second designer whose module planned to adopt filtering later, so a future-consumer perspective shaped the pattern before it shipped — not after it had already calcified around one team.

CHALLENGE

Accessibility as an inherited baseline. TOPdesk holds to WCAG, and filtering is interaction-dense. Anything I shipped became the accessible baseline for every module that consumed it.

APPROACH

Accessibility was reviewed with the specialists at the pattern level before shipping. Because modules consumed the components rather than rebuilding them, they inherited the keyboard model and semantics for free — the whole leverage of solving it once at the system level.

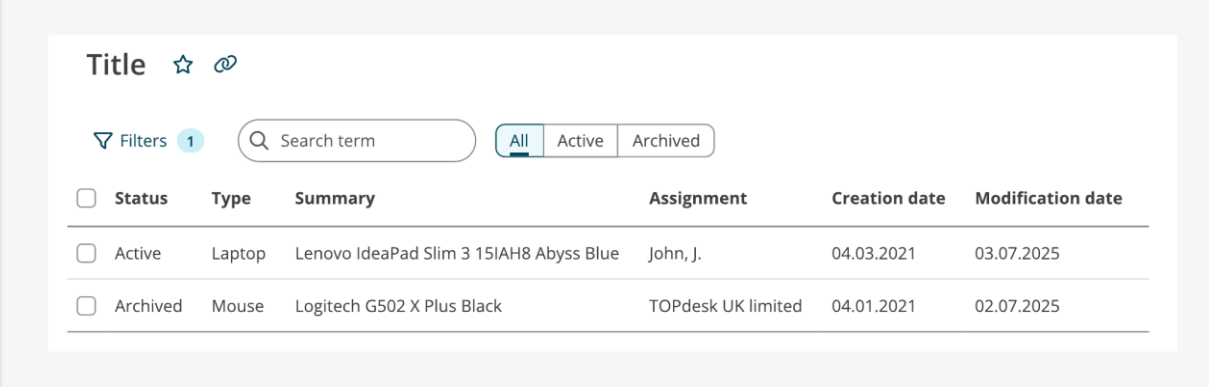
03 — THE PATTERN

Two locations, one model

The architecture is two places a filter can live, governed by one rule. Standard filters sit across the top of the dataset; the advanced panel hides at the side until opened. A filter lives in one or the other based on how often it's used and how complex it is — never both.

Standard filters — the top of the dataset

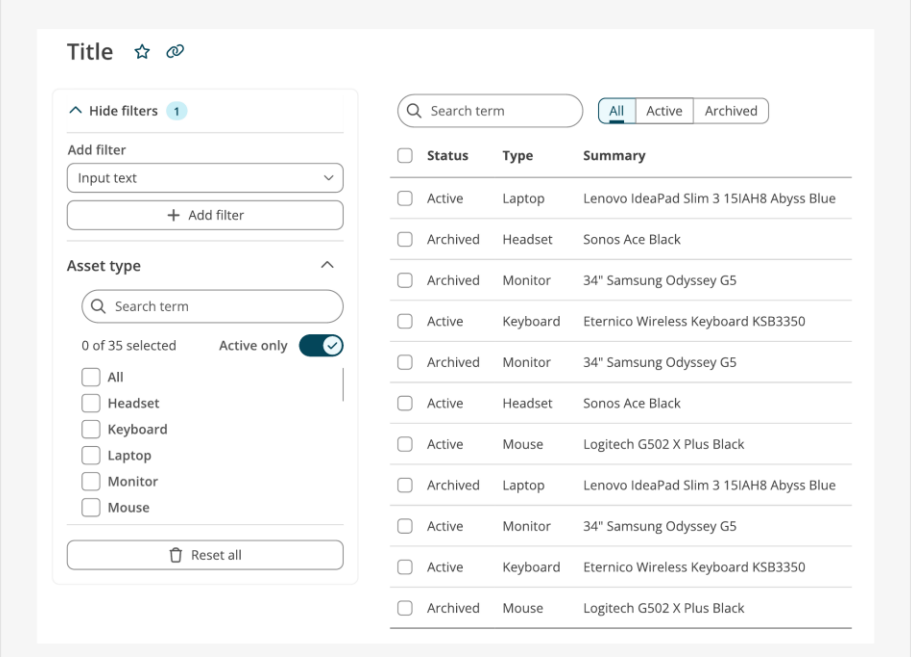
Always visible, low-effort, for the most frequent attributes — status, category, owner. Capped at about four controls so the bar stays fast to read; beyond that, filters move into the advanced panel. Four components cover it, in descending priority: **switch** for a binary show/hide, **toggle button group** for three mutually exclusive states (All / Active / Archived), and the two new **single- and multi-select dropdowns** for lists of 3–50 options.



Standard filters across the top of the dataset — switch, toggle group, and the new single/multi-select dropdowns, capped at ~4 for legibility

Advanced filters — the side panel

For depth and specificity: multiple conditions, ranges, larger option sets, and — the key capability — **adding and removing filter criteria**. A combobox lets users pick a filter to add; it appears at the top of the set; any filter can be removed, or the whole set reset. This is the real "build your own filter" experience that Asset Management needed. Where adding isn't required, the same panel simplifies to a fixed set with a single Clear all.



The advanced panel — a default filter set plus an Add filter combobox; criteria can be added, removed, or reset, composing a filter set as long as the use case needs

Choosing the right control

The guideline turns all of this into a decision a designer can make in seconds — crossing the kind of filter against how many options it holds. That matrix is what stops teams defaulting to the complex component for everything.

Filter need	Few / fixed	Up to ~50	Thousands
Binary	Switch	—	—
Quick / preset	Toggle button group	—	—
Single-select	Toggle / dropdown	New single-select	Searchable-select (panel)
Multi-select	New multi-select	New multi-select	Searchable-select (panel)
Advanced / built	Advanced panel — radio, checkbox, listbox, date, and bespoke components		

The decision matrix at the heart of the guideline: filter type against data size. The bottom-left cell — lightweight multi-select — was the gap the new components filled.

04 — VALIDATION

Testing it with real users

The pattern wasn't shipped on conviction. I ran an unmoderated prototype study with **70 real end users** in Useberry — mostly application and service managers who live in the product — in two stages mapped to the two locations, one for each half of the architecture.

Stage 1 — standard filtering

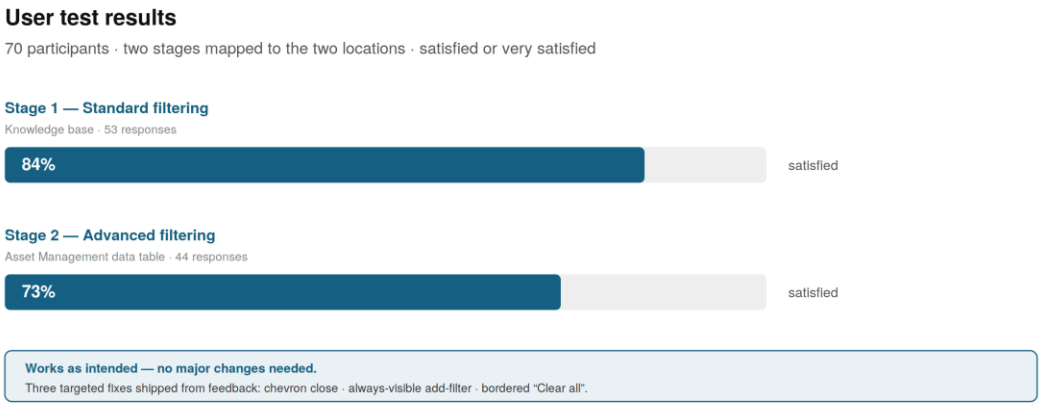
Using a knowledge base search, users were asked to narrow the results, exercising the new single- and multi-select dropdowns in the everyday case. It landed: of 53 responses, **84% were satisfied or very satisfied**, with only about one in ten negative. The open comments praised how quickly the mechanics were understood and how clear the UI was — exactly what the lightweight components were built to deliver. The one concrete fix it surfaced was small and sharp: the "Clear all" control looked like the filters themselves, so it shipped with a **border instead of as a ghost button** to set it apart.

Stage 2 — advanced filtering

In an Asset Management data table, users had to go beyond the defaults and **build a new filter criterion themselves**. The harder test, and it held up: of 44 responses, **73% were satisfied or very satisfied**. The friction it surfaced was specific and actionable — the two-step "Add filter, then choose the criterion" flow felt like one step too many, and the "X" used to close the panel read as "turn filtering off" rather than "hide."

What changed because of it

Both points went straight into the shipped pattern. The panel's close affordance became a **chevron instead of an "X"**, so it reads unambiguously as hide-and-show. And the add-filter flow was collapsed into an **always-visible combobox** — pick the filter and go, rather than pressing "Add filter" first and waiting for a dropdown. Together with the Clear-all fix from Stage 1, the test paid for itself: real friction points across both halves of the pattern, caught before build and resolved in the guideline.



What it confirmed. The headline finding was that the pattern works as intended — users understood it and used it easily, with no major changes needed, only the targeted refinements above. The most telling result wasn't in the ratings, though: asked what else would help, users **spontaneously asked for exactly what the pattern was built to deliver** — the same filtering from Asset Management in the incident module, the ability to filter from any overview, consistency everywhere. They were independently describing the global pattern, which was the strongest validation of the thesis I could have asked for.

05 — IMPACT

What changed

The honest version, for a pattern shipped in core modules and still rolling out when I left. I'm keeping it qualitative on purpose — I'd rather defend what I claim than dress it in numbers I didn't instrument.

Objectives

- Make filtering one learnable, cross-module behaviour instead of a per-module dialect.
- Close the component gap so multi-select no longer meant adopting the heaviest control in the system.
- Design a pattern every module could adopt — including those with no current use case — from one live one.

What it changed

- **A right-sized control for every case** — with the new 3–50 single- and multi-selects, teams stopped over-reaching for the complex searchable-select, and lightweight multi-select existed for the first time.
- **Consistency by construction** — the one-filter-one-location rule meant a user who could filter one surface could filter the rest, and never had to wonder which copy of a filter was real.
- **Decisions made faster** — the matrix turned "which filter component?" from a per-project debate into a lookup, which is what lets a pattern actually scale across teams.
- **A proven pattern layer** — filtering was the first large proof that the system's cross-module pattern layer was viable, not just the components beneath it. User testing backed it up: people spontaneously asked for this filtering everywhere in the product. That's the part I'm most confident outlives any single feature.

What I'm not claiming. The pattern itself is complete and shippable — the multi-select dropdown was finished before I left, so both halves work as designed. What it deliberately doesn't cover is **saving** — saved filters and saved overviews. We scoped that out on purpose: saving an overview is a larger pattern in its own right than persisting a filter, and folding it in would have muddied a filtering pattern that needed to ship clean. It was always meant to be a separate piece of work, likely in progress now — though having left, I can't confirm where it stands. Rollout across the remaining modules was also still underway: the design was done, the migration was the open question.

06 — COLLABORATION

How it got built across teams

A system-wide pattern is as much an organisational artefact as a design one. From the design-system seat, the job was less owning every pixel than making the pattern safe and easy for module teams to consume well.

- **The Asset Management designer** — my main collaborator, and the reason the advanced panel reflects a real live use case. I worked closely with them to understand how deep filtering actually got used day to day, so I was designing against reality, not a hypothesis.
- **The mobile / taskboard designer** — they wanted the pattern for the new taskboard they were building, so I brought them in early to shape it before it shipped. Designing for a surface with completely different constraints — far less screen space, touch instead of pointer — kept the pattern honest as a system pattern rather than letting it harden around Asset Management's desktop case alone.
- **Engineering and accessibility** — engineering owned technical architecture while I owned design architecture; the accessibility specialists reviewed at the pattern level so every consuming module inherited an accessible baseline.

The underrated lesson: the relationship between the system and the teams consuming it mattered more to adoption than the pattern's own polish. A pattern only goes global if the people who have to consume it want to.

Adam Pusztai
pusztaiad@gmail.com · linkedin.com/in/adampusztai