FieldOps Procurement Automation - Case Study

n8n FDE Case Study: Part 1 Submission

Design approach

When I design a system I am solving a problem, and most businesses share the same problems. A problem can be specific to an industry, a department, or a niche, but it is rarely unique to a single company. So the first thing I ask on any engagement is what the core problem is that they are trying to solve. If I build from the ground up around that problem, the workflow ends up transportable across other companies in the same industry.

1a: Pre-engagement planning

I start every engagement with an AI-led self-discovery, run before the first call. It is the most effective way I have found to understand how a process actually works. When people answer at their own pace, on their own time, sitting with their thoughts and not under the pressure of a 60-minute meeting, they surface far more than they ever would live. Running it ahead of time also lets me pull the data down first, so the call is not the start of discovery. I gather the facts and use the hour to confirm them.

The pre-discovery interview. Seven days before the call I send it to every stakeholder and everyone involved in the process. I also ask them to watch the process over the days leading up to the call and report what they see.

The interview asks each department the same core questions:

1. Where does the data live, and in what systems? Where are steps entered into systems we can pull logs from? Does the data need auditing or cleaning?
2. Where are the touch points in this process?
3. Do you have written, established rules for this department, and if so, where does that documentation live today?
4. Which department would give us the most immediate impact if we started there?
5. When an approval stalls today, who notices, how, and how long does it take?
6. Where does "approved" legally live today: a page, an inbox, or someone's memory?

For this engagement I also ask for the threshold table and approver list for their top candidate department, and for what is reachable from n8n today (mail, SSO, ERP) and who can grant credentials Monday morning. Those two answers decide whether five days is enough.

The call becomes a confirmation meeting. Because the interview already surfaced the findings, the discovery call is no longer discovery. It is where I confirm the findings and get the stakeholders to agree on them. Here is what your departments told me, here is where they disagree with each other and with your own data, and here is the one department I would start with.

Start with the busiest department. I pick the busiest, most active department on purpose, because I want to stress-test the system from the start and find where it fails while there is still time to fix it. If you start with the easy one, everything looks fine until you reach the big one, and then things break at the worst possible point. Surfacing those problems early is what makes everything after it run smoothly.

Scope. Scope for me is one business unit, working end to end, with regression tests, a dev sandbox, error reporting, and full handoff documentation. The workflows are built around the data sources, so

adding the other business units later is a matter of updating three tables, not touching the workflow logic. The eight-BU ask came from the Head of Digital Operations, so I settle the reframe with them before day 1: one BU built end to end, plus a rollout path their team runs for the other seven.

Roadmap. A few things are deliberately left for after the pilot. Automated RFQ sourcing, so an item with no approved vendor kicks off a request for quotes on its own instead of stopping. ERP integration, so an approved request writes back to the ERP and the purchase order is created there. And a dashboard to review everything in one place: the stats, the errors, and the requests, approvals, and rejections. None of these are needed to prove the pilot works, and each one depends on a system or a decision that is theirs to make, so they come after the first unit is live.

Baseline. I reconstruct a baseline from the touch points that already exist, and I would work with the IT department to pull that together ahead of time. If a request goes out as an email, gets logged in a Confluence tracker, and ends in a PO, I already have a rough audit trail I can rebuild. The question I ask is whether there is any other electronic touch point between the request and the log, any other email or system entry, because each one I find makes the trail more concrete. I anchor it with a short recall session with the pilot approver and get the CFO to sign it on day 1. The day-1 baseline does not have to be precise. What matters is that everyone agrees on it going in.

1b: Solution design

The design follows from one decision: build around the data. The rules that change between business units live in the data sources the platform team already owns, not in IF nodes on the canvas. For the demo I use n8n Data Tables to stand in for those sources, but in production the workflow reads from the customer's own systems. Three tables carry it: the business units, the approval rules (threshold band, primary and fallback approver, escalation window, auto-approve limit, cost center), and the vendor

catalogs. The workflow logic reads those sources. It never hard-codes a unit or a threshold.

Adding the other seven BUs is then mostly data entry. The eight units use different thresholds and approvers, but the process is the same shape, so one workflow plus config rows covers all of them. Changing an approver means editing a row. Nobody has to open n8n when a requirement changes.

The second decision: the workflow works out the amount itself. The requester says what they need, the workflow prices it against the approved vendor catalogs, and that price drives the routing. If the requester just types a number, you are approving a figure nobody checked, and the approval means nothing.

The pilot is one business unit, one category (indirect/MRO), and three vendor catalogs, built end to end. A request comes in through a form or an email and is normalized to one object. The workflow prices it against the approved catalogs, checks it against the unit's rules, and routes it: auto-approved when it clears the bar, or sent to a human for a single or dual sign-off on a Slack card that works from a phone, with a timeout that escalates to the fallback approver. A rejection sends the request back for revision instead of ending it. Every request writes one outcome row with its timestamps and decision. That row is what 1c measures against.

A few guardrails hold it together. If a unit has no rule, the request goes to a person instead of sliding through. If an item matches no catalog, it goes to sourcing so nothing gets approved without a price. And if Slack goes down, the approval still holds, because it lives in the workflow's own state and does not depend on Slack being up. Slack is just the channel I had on hand; the same notifications point at Teams, Google Chat, WhatsApp, or whatever the customer runs by swapping the send node. A separate monitor watches every workflow and posts to Slack and email the moment any run fails, so a failure surfaces on its own instead of hiding as a stalled request. The build is handed off to run without me, on the dev sandbox, regression suite, documented canvas, and private repo described in 1a.

1c: Measuring value

I measure every electronic touch point I can capture: when a request was submitted, when it was approved, when the PO got created, how long that took, and where it hung up. Then I look for any other touch point in the process, any other email or system entry, and build a forensic trail from them so the numbers are concrete instead of guessed. Any touch point, any stall point, anywhere I can prove exactly where a request stopped.

Without that trail you are guessing. A request stalls three days on an approval and the reason is ordinary. People are busy, everyone is asked to do more with less, someone is out sick, and it slips through the cracks. Capturing every touch point closes those gaps.

Everything is measured off one table called `pr_events`, with no separate dashboard to build. Each request writes a single row when it finishes, carrying its route, the price the workflow worked out, whether it went off-contract, the savings against what was asked for, the catalog-match result, why it routed the way it did, and its submitted and decided timestamps with the elapsed time. You get the measurement for free just by running the workflow. The only thing I hold firm on is using the same definitions before and after, so the comparison is real.

What I measure, and when:

When	What	How
Before (day 1)	volume per week, median and p90 cycle time, stall rate, approver touches	Reconstructed from the tracker's page history and the mailbox, triangulated with the interviews, signed by the CFO on day 1.
During (day 2 on)	one outcome row per request	The workflow emits the row. No extra instrumentation.
After (30/60/90)	the same metrics, the same query	Run against `pr_events`.

The workflow works out the price itself, so the requester's guess never drives anything. That one row hands the CFO the two numbers he cares about most, and both are real measurements. The first is how often requests were going to a vendor who did not have the best contracted price. The second is the actual euro gap between the vendor they asked for and the best one, on every approved request. I am upfront about the before: the old tracker only named the vendor about half the time, so the historical number is rough and I say so. From day 1 on it is exact.

The numbers are only half of it. At 30, 60, and 90 days I also send a short feedback survey to the stakeholders using the automation, so I hear directly from the people running it whether it is making their job easier and where it is getting in the way. The metrics tell me what happened. The survey tells me how it feels to the people using it, and it catches problems the numbers miss. That feedback goes straight back to the engineering team so the next round of changes is driven by what the people using it are actually saying.

There is also a monitor watching the workflows the whole time. Every few minutes it checks for any run that failed, and the moment one does it sends a Slack message and an email, so we know right away instead of finding out later when a request is stuck. Through the pilot and the full 90 days, a failure gets caught the same day it happens.

The 90-day story. By 90 days their team has moved one or more business units onto this, and we can see every step: how long each one takes, where things used to get stuck, and whether orders are actually arriving on time.

And I am straight with him about what the pilot did. It did not save him money yet. It found the number. Now he knows that Z% of his MRO requests were going to the wrong vendor, and he can take that rate across his other seven business units and work out what it is worth against his own volumes. I will walk him through how I got there. The thing I actually care about by then is how many of these his own team has built without me, because that is what he was paying for.

1d: Reusability and feedback loop

Most of what I build is templatable, and it is templatable because I start from that perspective and architect around the data. Plug in the different data systems, make a few node customizations for the schema, and as long as I am only editing the schema while the logic stays the same, it becomes a template. That is how I build all of my work. Business problems are not unique. The data inside them is unique, the problems are not.

So the reusable output is not a clever piece here and there. It is the whole build. Because it is architected around the data, the entire package, the workflow, the config tables, the tests, and the docs, stands up for the next customer by plugging in their data and adjusting the schema. The engagement produces one reusable package.

Feedback to Product and Engineering. Same approach as the baseline documentation I hand the client, adapted. I make the internal docs more specific about which parts are meant to change and which systems they plug into, and I go into more detail there for Product and Engineering. Anything I run into while building that the platform could handle better gets written up against the real case that exposed it, with the steps to reproduce it, and filed where they will see it. The 30/60/90 stakeholder feedback from 1c feeds the same channel, so the changes engineering makes are driven by what the people using it are telling me.

The internal playbook. A GitHub repo with the entire package we just built, sanitized and templated, ready to stand up for the next client in half the time. The pieces that carry across almost every engagement are the retro-baseline recipe, the discovery kit and its questions, and the golden-set test approach.