Case Study - Inbound Document Automation

Client-safe version. Safe to attach to any Upwork proposal.

Source work performed for a UK food and beverage supplier selling into large national grocery retailers. Client name, retailer names, endpoints, identifiers and all credentials are omitted by design. Every figure below is from production data, not estimates.

The problem

A shared inbox carried every commercial document the business received: purchase orders in a dozen different layouts, plus order confirmations, delivery notes, invoices, statements, stock reports, shortage claims and EDI relay traffic, all mixed together. Staff read each email, decided what it was, typed the line items into the ERP, and chased errors by hand.

Three things were true at the start, and they are the same three things every buyer of this kind of system says:

1. Accuracy was good enough until it was not, and the failures were invisible.
2. Volume growth meant headcount growth, because intake was the constraint.
3. Nobody could say how long the process actually took or what it cost.

What I built

One pipeline, five stages, all owned and operated by me. I built it incrementally on live traffic, never as a rewrite that took the business down.

Stage 1. Intake. Microsoft Graph change notifications on the shared mailbox. Each inbound message becomes one idempotent job in Postgres. No polling, no duplicates.

Stage 2. Classification. 5-way triage: order with attachment, order via portal, order in the email body, clearly not an order, or needs review. Deterministic sender and domain rules run first; the LLM is only consulted for genuine ambiguity. Prompt invariants are hard rules, not requests: identity (“we are the supplier, never the buyer”) and exclusion of automated confirmation reports.

Stage 3. Gating. Confirmations, invoices, statements and claims must never become orders. Each blocked document stores the marker evidence that justified the block, so a human can audit the decision rather than trust it.

Stage 4. Extraction, three tiers behind one output contract.

TIER	PATH	COST	USED WHEN
1	Config-driven deterministic engine. A database row per customer format holds sender pattern, filename regex, layout strategy and a per-field extraction strategy	$0	A template exists

TIER	PATH	COST	USED WHEN
2	Schema-constrained LLM with structured output, then the same validation gate	~$0.005/doc	No template, or low confidence
0	Vision model on the rendered page	higher	Scanned or broken text layer

Repeated Tier 2 success on a new sender auto-drafts a Tier 1 template. The expensive path shrinks over time by design. New customer formats are onboarded by inserting config, not by shipping code.

Stage 5. Validation, mapping and deployment. Quantity reconciliation against the document's own totals. Field completeness scoring. Price exception checks. Partner product codes resolved to internal SKUs through an exact-match, then fuzzy, then LLM tie-break ladder with confidence scoring; perfect matches skip the model entirely. Duplicate protection on the customer plus PO business key. Then a REST write to the ERP and on to the 3PL, behind a `disabled / shadow / live` gate. Shadow mode records the exact payload it would have sent and sends nothing.

The ops layer, which is the part that actually kept the engagement. A per-line evidence table that stores what the parser emitted for every single line, so extraction bugs are findable. A centralized execution log that pages into Slack. A daily health digest. A reconciliation sweep that answers the one question green dashboards cannot: *did every accepted document produce a record?* A human review queue with an approval status per record and a status tag written back to the source email, so ops can see state without opening anything.

Numbers

Documents processed through the pipeline	~5,900
Line items extracted and deployed	~9,900
Inbound triage decisions logged	2,289
Distinct partner document formats live	11+
Customer and product records mirrored	864 / 1,300
Active workflows on the instance	75 (of 179 total)
LLM cost per document	$0.005 - $0.02
Parser engine test suite	30/30 passing, byte-parity against legacy parsers on real orders
Orchestration refactor	65 nodes to 45, minus 31 percent, certified 40/40 green

How this turned into revenue, and into a bigger engagement

I will not dress this up as a percentage I cannot evidence. What actually happened, in order:

1. Volume stopped being a headcount problem. The pipeline runs 24/7. Retail orders land at odd hours; a dozen partner formats that used to need a person on a keyboard now need a person only when confidence is low. The business grew its order intake without growing the team that reads it. That is the direct margin line.

2. Speed became a commercial advantage, not just a convenience. In grocery supply, a purchase order that is keyed late is a delivery window that is missed, and a missed window is a rejected delivery, a penalty, or a lost slot. Automating intake compressed order-to-warehouse time and protected revenue that manual lag was quietly leaking.

3. Wrong prices stopped shipping. Price exception alerts caught cases where an order arrived at a price that did not match the agreed tier. Margin leakage is invisible until someone reconciles it. Someone now does it automatically.

4. New accounts stopped costing engineering time. Onboarding a new partner format used to mean duplicating an entire workflow and hard-coding a parser, one to two days each. With the config-driven template table it is a database row plus a regression check. Commercially that means the operations team can say yes to a new retail account without waiting on me.

5. The automation proved its own ROI, and that is what grew the spend. I built a telemetry layer that converts every workflow's successful executions into hours saved per department, computed as executions multiplied by the manual minutes that task used to take, synced nightly into a dashboard. Roughly 90 tracked workflows, 86 active. Because the savings were visible weekly, the work spread from one order pipeline into hiring automation, an IT support intake flow, a training hub, sales reporting sync, and monitoring. Scope grew because the numbers made the case for me.

6. My own rate and position changed because of it. From a small hourly contract to owning the entire automation estate, with a maintenance retainer and a proposed employee conversion on the table. Same work. The difference was evidence, documentation and reliability.

The lesson I sell now: **the automation market is full of people who can wire a model to a webhook. The value is in the failure path.** Every one of the revenue points above came from something that would otherwise have been a silent failure.

The three silent failures I found and fixed

Worth quoting in an interview, because every buyer of document automation has these and cannot see them.

1. **The rubber-stamp classifier.** An automated "orders confirmed" report from a warehouse was classified as an order because it contained the word "orders" and an attachment. It then hallucinated a *different real customer* as the buyer through an address fallback. Fixed with identity invariants, EDI-report exclusion, and by rewiring dead-end branches that had been stalling batch verification for every email in the batch.
2. **The daily crash.** A claims mailbox spreadsheet was routed to the extraction agent despite an explicit "ignore this sender" rule, because the downstream gate only asked "is the customer known?" and never "is this even an order source?". The agent correctly refused, and the fallback node then hung the task runner for 300 seconds. Same email, three failed runs, every day.

3. **The first-item-only bug.** A mapping loop processed only the first SKU of every multi-line order. For months. Found because I had built the per-line evidence table specifically so parser output would be auditable. That is the whole argument for instrumenting extraction.

And the one that made me a convert to reconciliation checks: a change made an extractor return the wrong payload shape to its caller. Emails arrived, were classified ready, were parsed correctly, and produced no orders for two days. Every execution was green. No alert fired. It was found by accident. Twenty real order documents produced nothing. After that I never shipped a pipeline whose health depended on its own status codes.