

PhilGEPS Tender Response Automation

A case study in scraping, AI extraction, and retrieval-grounded drafting for Philippine government procurement

Overview

This is a personal automation project built to demonstrate how a small Philippine business could realistically monitor, evaluate, and respond to government procurement opportunities without hiring a dedicated bid team. I built it around a fictional client, BayaniTech Supplies, an Antipolo based IT equipment and services provider, but every piece of tender data the system processes is real and live, pulled directly from PhilGEPS, the Philippine government's official procurement portal.

The Problem

Small suppliers who want to bid on government contracts face a specific set of frictions. Tenders are published across more than a hundred categories on a portal with no public API. Each notice lists its own eligibility requirements, phrased differently from agency to agency, and missing even one document disqualifies a bid outright. Deadlines are often tight relative to how long a proper quotation takes to prepare. For a business without a full time procurement officer, the practical result is that good opportunities get missed, or bids get submitted with incomplete documentation.

What I Built

The system is structured as an eight stage pipeline: receive, normalize, enrich, decide, notify, escalate, and update, with routing logic built into the decide and enrich stages rather than existing as a separate step. The receive stage scrapes PhilGEPS directly. Normalize takes the raw tender text and uses an LLM to extract a consistent structured record, including the procuring entity, budget, closing date, and the specific eligibility documents required. Enrich compares that requirement list against BayaniTech's own compliance document library, stored in Airtable, and flags anything missing or expired. Decide scores the opportunity against BayaniTech's stated fit criteria, contract size, category match, compliance readiness, and days remaining before closing, and returns a go, no go, or borderline verdict with a written rationale. Notify sends that verdict to a Telegram bot in real time. Escalate recalculates the time remaining before closing and fires an urgent follow up alert if a tender is inside a 48 hour window. Update is the most involved stage: on approval, the system retrieves the most relevant facts about BayaniTech from a small vector knowledge base and uses them to draft an actual quotation response letter, grounded in real company information rather than generic filler text.

Technical Approach

The hardest part of this build had nothing to do with AI. PhilGEPS is a legacy ASP.NET WebForms site with no API, which means every navigation action, clicking into a category, moving to a results page, is handled through a server side postback mechanism rather than a normal URL. Reproducing that in n8n meant capturing the page's `__VIEWSTATE`, `__VIEWSTATEGENERATOR`, and `__EVENTVALIDATION` tokens along with session cookies on every single request, since ASP.NET invalidates stale tokens and each new page returns a fresh set that has to be threaded into the next request. Getting this right took a lot of trial and error, including discovering that the site's own JavaScript sets a hidden form field before submission that isn't obvious from the page source alone. For the drafting stage, I wanted the generated proposal language to be grounded in real facts about the company rather than left to the model to invent. I built a small retrieval pipeline using Qdrant as a vector store, with company facts embedded using Hugging Face's free inference API rather than OpenAI's, to keep the whole stack free to run. When a tender comes in, the system embeds a summary of it, searches the knowledge base for the most relevant facts, and only then generates the draft, which noticeably improved how specific and accurate the output was compared to giving the model the tender alone.

Challenges Along the Way

A few things broke in ways worth mentioning honestly. Hugging Face deprecated the embeddings endpoint I originally built against midway through, which meant tracking down and switching to their new router based URL structure. Several n8n specific quirks cost real debugging time too: Code nodes using `$input.first()` silently process only the first item in a batch unless you explicitly loop with `$input.all()`, and Qdrant rejects numeric or array fields if they're wrapped in quotes inside the request body, which isn't obvious from the error messages alone. I also hit a real limitation with n8n's built in looping node, which doesn't reliably support open ended manual recursion, and ended up handling multi page pagination through workflow static data instead.

Current Status

All eight stages have been built and individually verified against live PhilGEPS data, not mocked examples, using a real tender from the Bureau of Jail Management and Penology as the test case throughout. The remaining work is consolidation: wiring the pipeline so it runs on a schedule and processes every tender in a category automatically, rather than being tested stage by stage against one tender at a time, and upgrading the Telegram notification into a proper approval gate so a reply from me is what triggers the drafting stage.

Tech Stack

n8n for orchestration, self hosted. Airtable for the compliance document library. Qdrant Cloud for vector storage. Hugging Face's free inference API for embeddings. OpenAI for extraction, scoring, and drafting. Telegram Bot API for notifications and escalation alerts.