CASE STUDY · 2026

EuFund

AI-Powered EU Funding & Project Management Platform

A multi-agent system that turns the fragmented, paperwork-heavy EU funding landscape into a deterministic, auditable workflow — from discovery to proposal to compliance review.

Author: Vasile Bogdan Godja
Role: Founder, sole engineer
Stack: Next.js 14 · TypeScript · PostgreSQL 16 + RLS · Drizzle · Qdrant · Redis · Docker · Google Cloud Run
LLM layer: Claude · OpenAI · Gemini · MCP · Vercel AI SDK, with cost-aware routing
Status: Live deployment · Romanian market focus · open source at github.com/Bogdan0708/EuFund
Contact: godjabogdan@gmail.com · github.com/Bogdan0708

---

01 The Problem

SMEs and municipalities across the EU are leaving billions in funding on the table — not because the money isn't available, but because the process of finding, qualifying, and applying for it is structurally hostile to anyone without a full-time consultant.

What the user actually faces

- **Fragmented discovery.** EU, national, regional, and sectoral programmes live on separate portals, with no shared schema and inconsistent eligibility language.
- **Opaque eligibility.** "Eligible activities" sections are dense, jargon-heavy, and frequently contradict the call's annexes. Mistakes here invalidate months of work.
- **Manual proposal drafting.** Each programme has its own template, narrative voice, and required annexes. Reusing prior material requires constant re-formatting.
- **Compliance theatre.** Even successful applications often fail audit because supporting documentation drifts from the approved budget over the project lifecycle.
- **No feedback loop.** Applicants rarely learn *why* they were rejected, so they keep making the same structural mistakes.

Why this matters to me personally

Before building EuFund I spent 6+ years as a general manager of a Romanian restoration company delivering on 20+ UNESCO and heritage-listed sites, and personally secured and managed **€500K+ in EU grant funding** — eligibility, application, documentation, implementation, and audit. Earlier still, I worked as a financial consultant on bank loans and EU grants. I have lived inside this workflow as a customer for over a decade.

> *The single most valuable input to EuFund's design was not market research — it was knowing, from direct experience, where the workflow actually breaks.*

Why existing tools fail

Government portals

Built for record-keeping, not for application work. Search is keyword-only, eligibility logic is buried in PDFs, and there is no path from a programme to a draft.

Consultancies

Expensive (5–15% of grant value), capacity-constrained, opaque about success rates, and disincentivised from telling clients when a programme is a poor fit.

Generic LLM chats

Hallucinate eligibility, can't cite, lose state across sessions, and produce plausible prose with zero audit trail — the worst possible failure mode for compliance work.

Workflow tools (n8n, Zapier)

Strong as glue, but provide no domain reasoning. Without a model that understands eligibility and proposal structure, they automate the wrong layer.

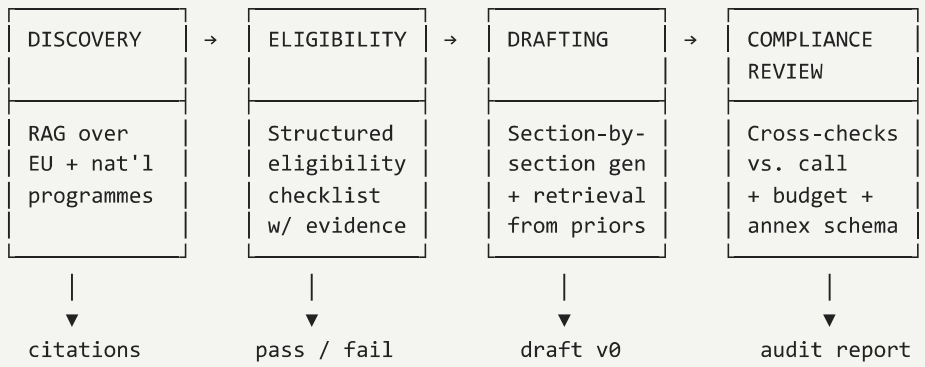
The opportunity

The EU funding stack is exactly the kind of problem applied AI is built for: messy long-form documents, repeatable structural patterns, multi-step reasoning, and a high tolerance for "draft" work as long as the audit trail is intact. The right system replaces a consultant's first 80% of work — discovery, eligibility, scoping, first-draft proposal — and lets a human expert focus on the high-judgement 20% that actually moves a win-rate.

02 The Solution

EuFund is a multi-agent platform that takes a user from "I want funding for project X" to a citable, audit-ready proposal draft — without ever leaving the application.

The four-stage workflow



Each stage produces an artefact that the next stage consumes, with the user able to intervene at every boundary. Critically, every output is structured and citable — no free-text answers, no orphaned claims.

Product surface

Discovery

Project-shaped intake (sector, region, budget band, timeline). Returns a ranked list of programmes with eligibility flags, deadline urgency, and a "fit score" with reasoning.

Eligibility

For each shortlisted programme, a structured checklist (legal form, geography, sector, activities, financial thresholds) is filled in and evidenced from the call's source text.

Drafting

Section-by-section proposal generation guided by the call's template, with retrieval from the user's prior projects, organisation profile, and approved budget components.

Compliance review

Pre-submission audit. Cross-references draft narrative against budget, annexes, and call requirements. Flags inconsistencies before submission rather than after.

Key product decisions

- **Deterministic workflow, LLM-driven reasoning.** The pipeline is a state machine, not a free-roaming agent. The LLM decides *content*; the system controls *flow*.
- **Citation enforcement at the schema level.** No claim about a programme ever ships without a span pointer to the source document.
- **Structured outputs everywhere.** Eligibility, scoring, and draft sections are all JSON-schema-validated. Free-text outputs only appear in the rendered UI.
- **Human-in-the-loop by default.** Every stage has an explicit accept/edit boundary. The system is opinionated but never silent.
- **Cost-aware model routing.** Different stages route to different providers based on task profile (cheap for classification, expensive for long-form drafting).

What we deliberately did *not* build

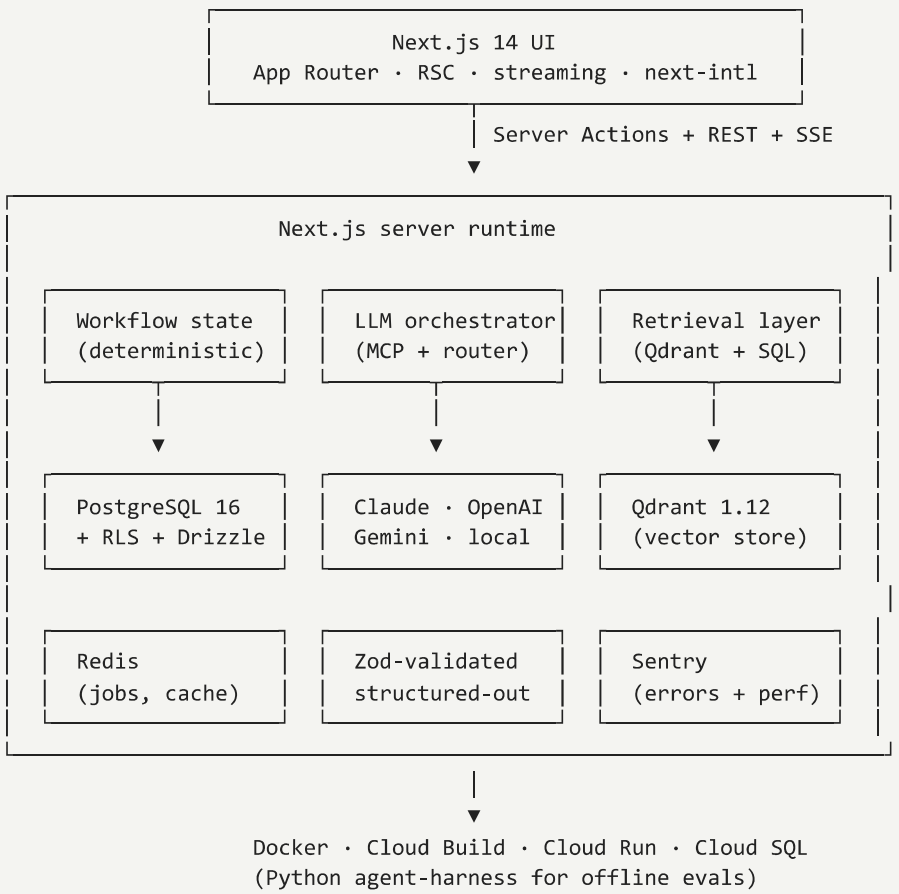
- No autonomous "agents that submit on your behalf." Submission is a human action; we draft, the user submits.
- No bespoke fine-tuning. RAG + structured prompting + provider rotation outperformed early fine-tuning experiments and removed an entire ops surface.

- No "chat with your funding programme" feature. Chat felt productive in demos and produced fewer applications in practice.

03 Architecture

EuFund is a server-first Next.js 14 application with three distinctive layers: a deterministic workflow spine, a multi-provider LLM orchestrator (MCP-exposed), and a retrieval layer built on Postgres + Qdrant with Row-Level Security as the multi-tenancy enforcement primitive. Boring infra, opinionated AI surface.

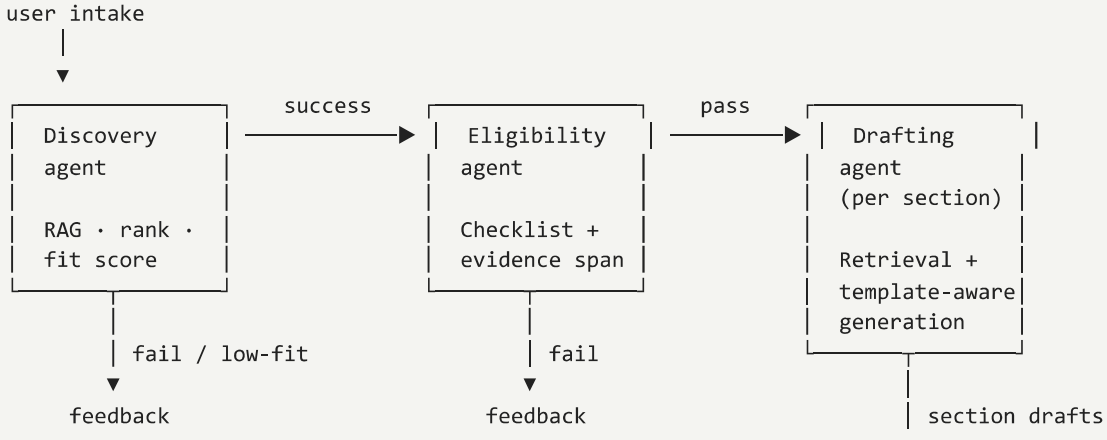
High-level system diagram



Stack

Next.js 14 (App Router) · TypeScript · Tailwind · shadcn/ui (Radix) · next-auth v5 · next-intl (RO + EN) · PostgreSQL 16 · Row-Level Security · Drizzle ORM · Qdrant 1.12 · Redis 7 (ioredis) · Vercel AI SDK · MCP · Claude (Anthropic) · OpenAI · Gemini · Local LLMs (Ollama) · Zod · Sentry · Vitest · Playwright · Docker · Cloud Build / Cloud Run · Python (offline evals harness)

Multi-agent orchestration in detail



Multi-provider LLM routing

The streaming surface uses the Vercel AI SDK (`ai` / `@ai-sdk/react`); the deeper orchestration is plain Next.js server actions calling a custom router so providers can be swapped per stage without rewriting frontend code. Tool use is exposed over the Model Context Protocol — the same tool surface can be invoked by the in-house orchestrator or by external clients (e.g. Claude Desktop) when needed.

Stage	Default provider	Why	Fallback
Discovery ranking	Gemini Flash / local	Cheap classification at scale	OpenAI mini
Eligibility extraction	Claude Sonnet	Best citation behaviour over long calls	OpenAI o-series
Section drafting	Claude Opus / GPT-5	Long-context, voice control, instruction-following	Cross-provider
Compliance cross-check	OpenAI structured-out	Zod schema reliability at low temperature	Claude with retry
Embeddings	OpenAI text-embed-3	Cost / quality sweet spot	Local BGE for sensitive corpora

RAG pipeline on Qdrant + Postgres

1. **Ingestion.** Programme PDFs and HTML are parsed (`pdf-parse`, `cheerio`) and chunked with structural awareness — preserving call sections, annexes, eligibility tables. Each chunk carries provenance metadata: programme ID, section type, effective date, source span.
2. **Indexing.** Qdrant with payload filtering and named vector spaces per stage — eligibility and drafting use different embedding profiles. Postgres holds the canonical document state; Qdrant holds the search index.
3. **Retrieval.** Two-stage: payload-filtered recall in Qdrant, then a small reranker pass. Top-k is stage-specific (eligibility wants precision; drafting wants breadth). Filters (section type, effective date, programme ID) run at the vector store, not as post-filter.
4. **Citation enforcement.** Every model claim is required to ship with a chunk ID. Outputs without citations are rejected by the Zod validator and re-prompted with the failure appended.
5. **Hallucination mitigation.** Structured-output schemas force the model to either cite or explicitly say "not in source." Free-text answers are not allowed where a citation is expected.

04 Technical Challenges

The interesting engineering in EuFund wasn't picking a model — it was the work between the model calls: workflow state, retrieval quality, evaluation, and cost.

1. Reliability at the workflow boundary

The first version used a free-roaming agent loop with tool use. It worked in demos and failed unpredictably in real flows — sometimes skipping eligibility, sometimes drafting against the wrong programme. The fix was to invert control: workflow state lives in Postgres as an explicit state machine, and the LLM is a tool the state machine calls, not an orchestrator. Reliability went from "uncomfortable" to "boring."

2. Retrieval quality on long, structurally heterogeneous documents

EU programme calls are not articles. They're 60–120 page PDFs with eligibility tables, annexes, and recursive cross-references. Naive chunking either split eligibility tables in half or buried relevant rows in low-similarity neighbourhoods.

The solution was section-aware chunking with structural metadata, plus payload-filtered retrieval in Qdrant using section-type and effective-date as filters at the vector store rather than as post-filters. Eligibility queries only retrieve from eligibility sections; drafting queries pull from objectives, deliverables, and prior approved projects; regulation queries respect "currently in force." Different stages use different named vector spaces — eligibility and drafting have different embedding profiles because they answer fundamentally different questions. Recall on the internal evaluation set improved materially without changing the model.

3. Structured outputs vs. expressive prose

Eligibility needs to be a checklist with evidence spans. Proposal drafting needs to be natural, persuasive prose. The same model can do both, but only if the contract is enforced per call. EuFund uses Pydantic schemas for every structured stage and reserves free-form prose for explicit "render section" calls — and even those are wrapped with pre/post-conditions (length bounds, required headings, no orphan claims).

4. Multi-provider routing without operational chaos

Routing across providers is easy to talk about and easy to break in production. The discipline that mattered: every stage declares a primary provider and an explicit fallback; every call goes through one adapter that normalises retries, token accounting, and structured-output validation (Zod schemas, validator-driven retry with the failure appended). When a provider degrades, the adapter retries on the fallback transparently. Tools are exposed via MCP so the same surface is callable from the in-house orchestrator or from external clients without duplicating wiring.

5. Tenant isolation at the database layer

EU funding work runs against sensitive applicant data — organisation profiles, financials, IDs. Tenant isolation in application code is one untested branch away from a leak. EuFund enforces tenancy with Postgres Row-Level Security: every query carries the tenant context in the session, and the database refuses to return rows that don't match. The application layer cannot bypass it because the database doesn't allow it. This is the kind of decision that costs a day up front and saves a year of audit theatre.

6. Cost

Per-proposal cost matters because the unit economics of grant work are real. The discovery and eligibility stages run dozens of LLM calls per session; drafting runs fewer but bigger ones. The discipline was: cheapest viable model per stage, aggressive caching of structural extractions (a programme's eligibility doesn't change between users), and reuse of embeddings across users where corpora are public.

7. Evals

EuFund's eval suite lives in a separate Python harness (`app/agent-harness`) so model evaluation doesn't pollute the runtime. It covers three slices: retrieval quality (recall@k on labelled eligibility questions), structured-output schema pass

rate per provider, and end-to-end "did the draft pass compliance review" — the last run with LLM-as-judge using a different provider from the one that produced the draft. Unit tests live in Vitest, end-to-end flows in Playwright.

8. Deployment and observability

Containerised with a multi-stage Dockerfile, deployed via Cloud Build to Cloud Run against Cloud SQL (Postgres) and Memorystore (Redis). Qdrant runs as a separate managed instance. Sentry captures errors and performance traces. Every LLM call is logged with provider, latency, token counts, cost, and the workflow step that produced it. When something goes wrong, the question is rarely "what did the model say" — it's "which workflow step routed badly, and why."

Failure mode	Mitigation
Hallucinated eligibility	Citation enforcement; outputs without source spans rejected at Zod validator
Schema drift in structured outputs	Zod-validated adapter with retry-on-error including validator message
Provider outage	Per-stage primary + fallback declared in config, transparent retry in adapter
Retrieval misses on long PDFs	Section-aware chunking + Qdrant payload filtering + named vector spaces per stage
Workflow drift in agent loops	State machine in Postgres; LLM is a tool the FSM calls, not an orchestrator
Tenant data leak	Postgres Row-Level Security enforced at DB layer; app code cannot bypass
Runaway cost	Per-stage provider routing, cached structural extractions, embedding reuse

05 Business Impact & Lessons

Where EuFund creates value

€500K+	4-stage	4×	RLS
EU grants I personally managed pre-EuFund	Deterministic pipeline replacing free-roam agents	LLM providers with cost-aware fallback	Tenant isolation enforced at the database layer

Who EuFund is for

- **SMEs** applying for innovation, R&D, and digitalisation funding (Horizon, EIC, regional ERDF).
- **Municipalities** applying for infrastructure, social, and green-transition programmes.
- **Consultants** who currently hand-build proposals and could 3–5× their capacity per associate.

Scale-out story

Romania is the launch market because of regulatory familiarity and because the funding workflow there is unusually paper-heavy — i.e. the value of automation is highest. The architecture is country-agnostic; expansion is primarily a content problem (ingesting a new country's programme corpus) rather than a code problem.

Lessons learned

1. Workflow state beats agent autonomy for regulated workflows.

Free-roaming agents are seductive in demos and unsafe in production. A deterministic state machine that calls the LLM as a tool is faster to ship, easier to debug, and the only thing an auditor will accept.

2. Structure beats prompting.

Pydantic schemas and validator-driven retries do more for reliability than any prompt iteration. Every time I tried to "just prompt around" a structural problem, I rebuilt the structural fix six weeks later.

3. Citations are a feature, not a constraint.

Forcing the model to cite slows the per-call experience and ships a fundamentally more trustworthy product. In a compliance domain this is not optional — but it's also the feature users notice and trust first.

4. Multi-provider routing pays for itself the first time a provider degrades.

The cost is one well-designed adapter. The benefit is uptime, price optimisation, and optionality when a new model lands. This stops being a "nice to have" the first time your primary provider has a bad week.

5. The operator's job is to know which 20% of the workflow not to automate.

The strongest design instinct I brought to EuFund came from being the customer. The parts I refused to automate (submission, final narrative pass, budget logic) are the same parts a thoughtful operator would have refused to outsource. That distinction matters more than any model choice.

6. Boring infra is the unlock for opinionated AI.

Postgres with Row-Level Security, Drizzle for typed queries, Qdrant for retrieval, Zod for output validation, and a Vercel AI SDK + MCP split at the streaming surface — none of those are exotic. That's the point. The "interesting" work in EuFund is the workflow spine, the citation discipline, and the multi-provider routing. Every other decision is a deliberately conservative one so the interesting work stays interesting instead of getting buried by infra surprises.

What I'd do differently from day one

- Build the evaluation harness *before* the second stage of the pipeline, not after the fourth.
- Treat the Zod validator as a first-class infrastructure component, not a wrapper.
- Invest earlier in the multi-provider adapter — even at 2 providers, the abstraction pays off immediately.
- Write RLS policies alongside the schema, not as a hardening pass three sprints later.

- Write the architecture diagram before the third refactor, not after.

What this case study demonstrates

- End-to-end ownership of a non-trivial applied AI system, from problem framing to production deployment.
- Architecture decisions that hold up under hostile interview questioning (state vs. agents, retrieval design, structured outputs, multi-provider routing, evals).
- Operator-grade judgement about which parts of a workflow deserve automation, and why.

> *EuFund is the system I wish I'd had as a consultant ten years ago. That's the only test I needed to run before deciding to build it. The code is open source at **github.com/Bogdan0708/EuFund**; the README and this case study are intended to be read together.*