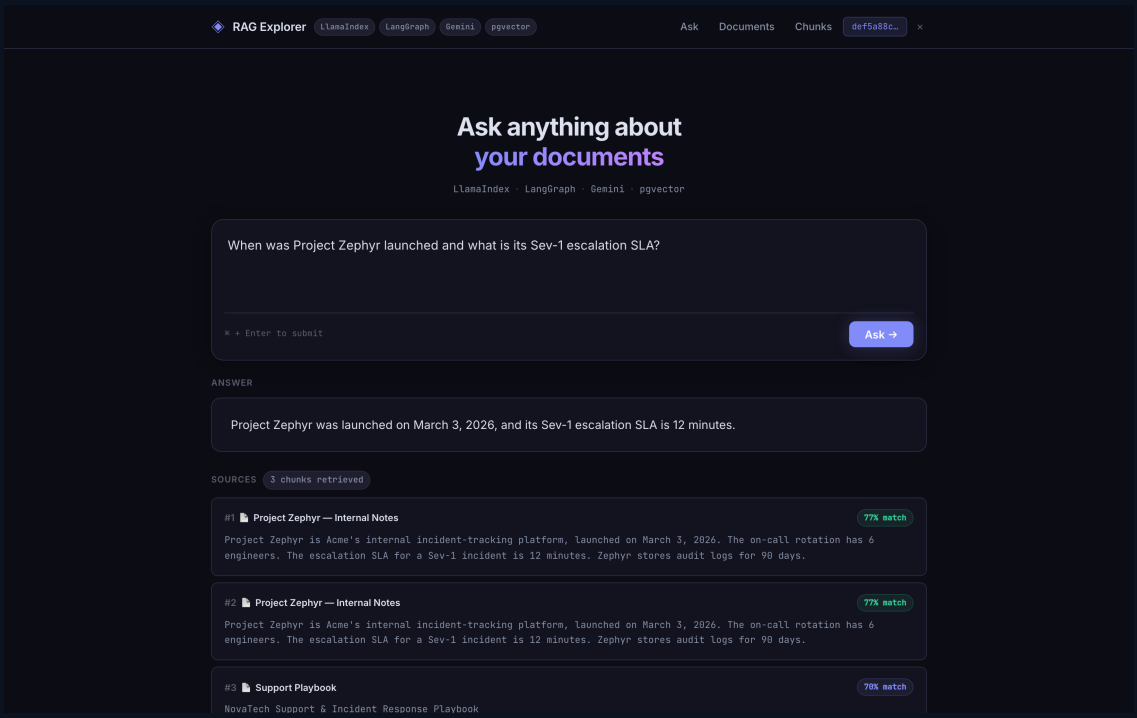
// portfolio

RAG Explorer

Semantic document search powered by LLM + vector embeddings



> Serhat Keskin

June 2026

> Contents

1 Problem & Context

Enterprise teams accumulate knowledge across hundreds of documents — product guides, HR policies, engineering runbooks, API references, support playbooks. Searching this content with keyword tools surfaces noise rather than answers. Teams waste time hunting through PDFs and wikis instead of getting direct, contextual responses.

The challenge is to build a system that:

- understands the *meaning* of a question, not just its keywords,
- retrieves the most relevant passages from a large document corpus,
- synthesises a coherent, grounded answer citing its sources, and
- remains fully auditable — showing which chunks drove the response and how confident the retrieval was.

```
$ Add: "Project Zephyr — Internal Notes" (your own document)
Ask: "When was Project Zephyr launched and its Sev-1 SLA?"
→ Grounded answer from your doc: launched Mar 3 2026, 12-minute SLA
```

2 Solution Overview

RAG Explorer is a production-deployed Retrieval-Augmented Generation (RAG) pipeline with a polished web interface. Users submit natural language questions; the system retrieves the most semantically relevant document chunks from a pgvector store and feeds them into Gemini to produce a grounded, cited answer.

Key design decisions:

- **Retrieval-first**: LlamaIndex manages chunking, embedding, and vector retrieval — ensuring answers are always grounded in indexed content.
- **Auditable sources**: every response includes the retrieved chunks, their relevance scores, and the source document — users can verify every claim.
- **Expiring demo tokens**: access is controlled via time-limited UUID tokens stored in Django, with per-user document isolation.
- **Edge-deployed frontend**: Next.js static export on Cloudflare Pages with a Pages Function API proxy — no CORS, global CDN, zero cold starts.
- **Self-hosted backend**: Django REST on a Proxmox VM behind Nginx Proxy Manager, containerised with Docker and Gunicorn.

3 Architecture

3.1 System Diagram

3.2 RAG Pipeline

The LangGraph state machine has two nodes:

1. **Retrieve**: LlamaIndex queries pgvector with the embedded question (`gemini-embedding-2`, 3072 dims). Returns top-3 chunks with cosine similarity scores. Each chunk header includes the source document title so the LLM has full context.

```
Browser (HTTPS)
↓ rag.projects.serhatkeskin.com
Cloudflare Pages (Next.js static export)
↓ /api/* → Pages Function (edge proxy)
↓ http://api.internal.example.com
Nginx Proxy Manager (NPM, 10.0.0.10)
↓ → 10.0.0.20:8000
Django + Gunicorn (Proxmox VM, Docker)
↓ LlamaIndex + LangGraph + Gemini API
pgvector (PostgreSQL 16 + pgvector extension)
```

2. **Generate**: Gemini receives the retrieved chunks and the question in a structured prompt. The response is plain text, concise, and grounded — the prompt explicitly forbids inventing facts not present in context.

```
$  $ curl -X POST /api/query/ -H "Authorization: Bearer {token}"
-d '{"query": "What are the pricing tiers?"}'
```

3.3 Token-Based Access Control

Demo tokens are UUID values stored in a Django `DemoToken` model with an `expires_at` field. The `DemoTokenMiddleware` validates every non-health request. Tokens are issued via a Django management command (`create_demo_token --label --days`) and can be revoked by setting `is_active=False`. Documents uploaded by a token holder are scoped to that token; the pre-loaded NovaTech demo data is visible to all.

4 Feature Walkthrough

This walkthrough follows a real first-time user end-to-end: they land unauthenticated, enter a demo access token through the UI, *add their own custom knowledge*, then ask a question about *that newly-indexed content* — not a pre-seeded example.

4.1 Landing — Unauthenticated

The app opens in an unauthenticated state. Example prompts are offered as a convenience, but this walkthrough ignores them — the goal is to bring *your own* data and query it.

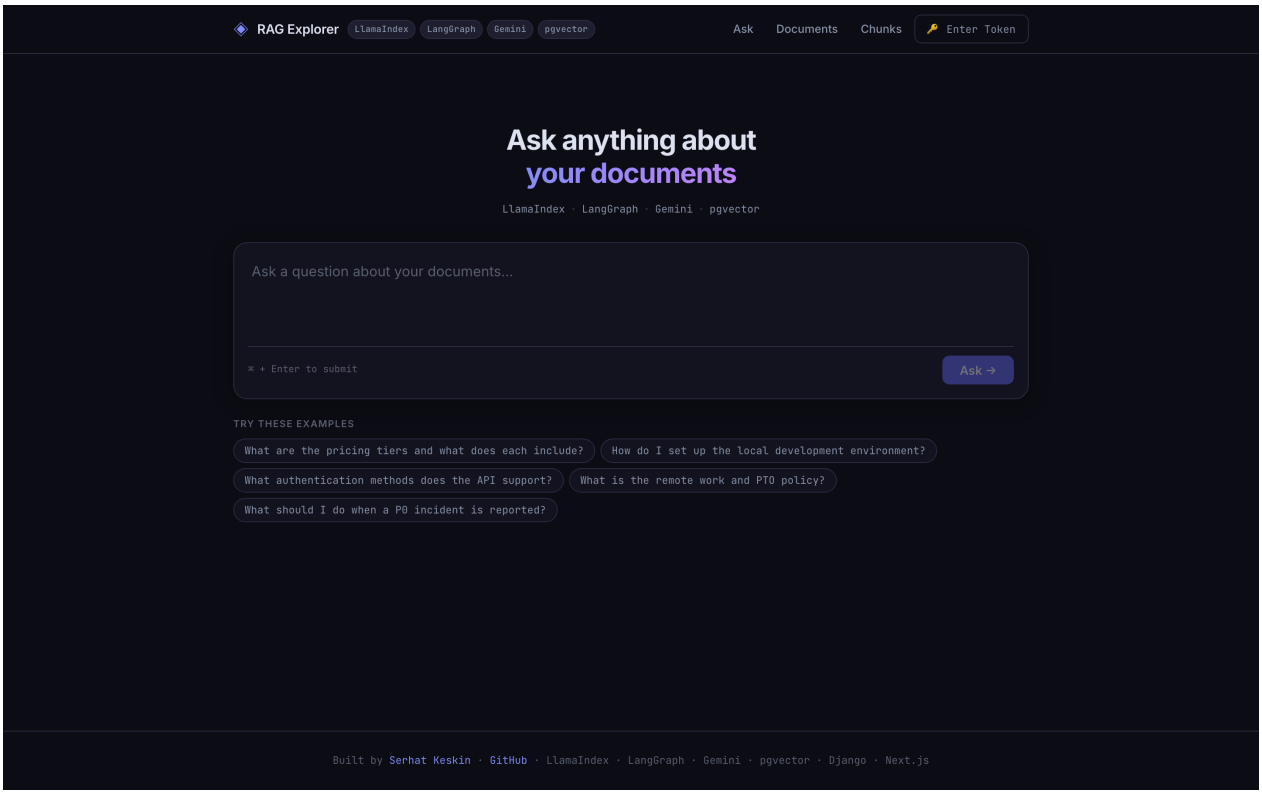


Figure 1 First visit — no token yet. The header shows an ”Enter Token” action; the query box is visible, but any query without a valid token returns 401.

4.2 Demo Access Token

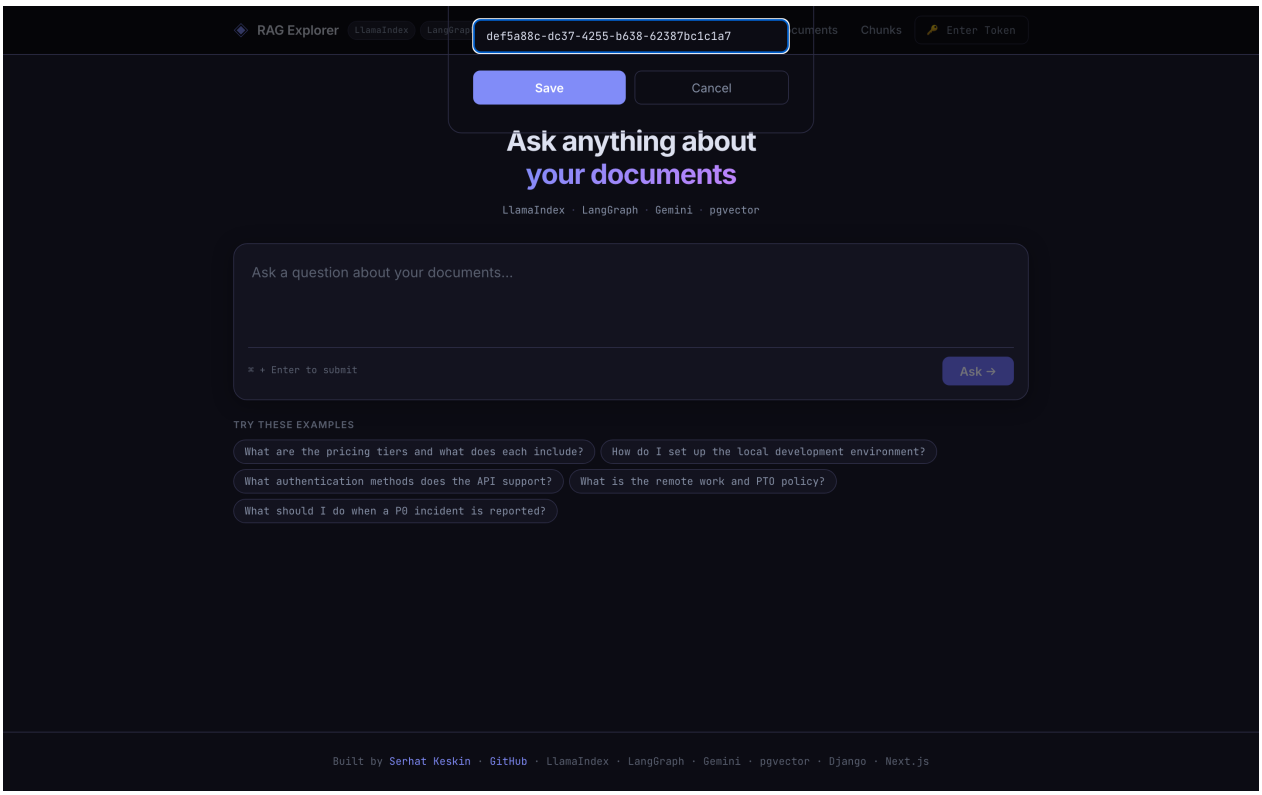


Figure 2 Token modal — the user pastes a time-limited UUID demo token. The format is validated client-side before it is stored and the session reloads as authenticated.

Access is controlled with expiring UUID tokens issued by Django. The token is validated against a regex, persisted in `localStorage`, and attached as a `Bearer` header on every API call — no signup, no OAuth flow.

4.3 Add Your Own Knowledge

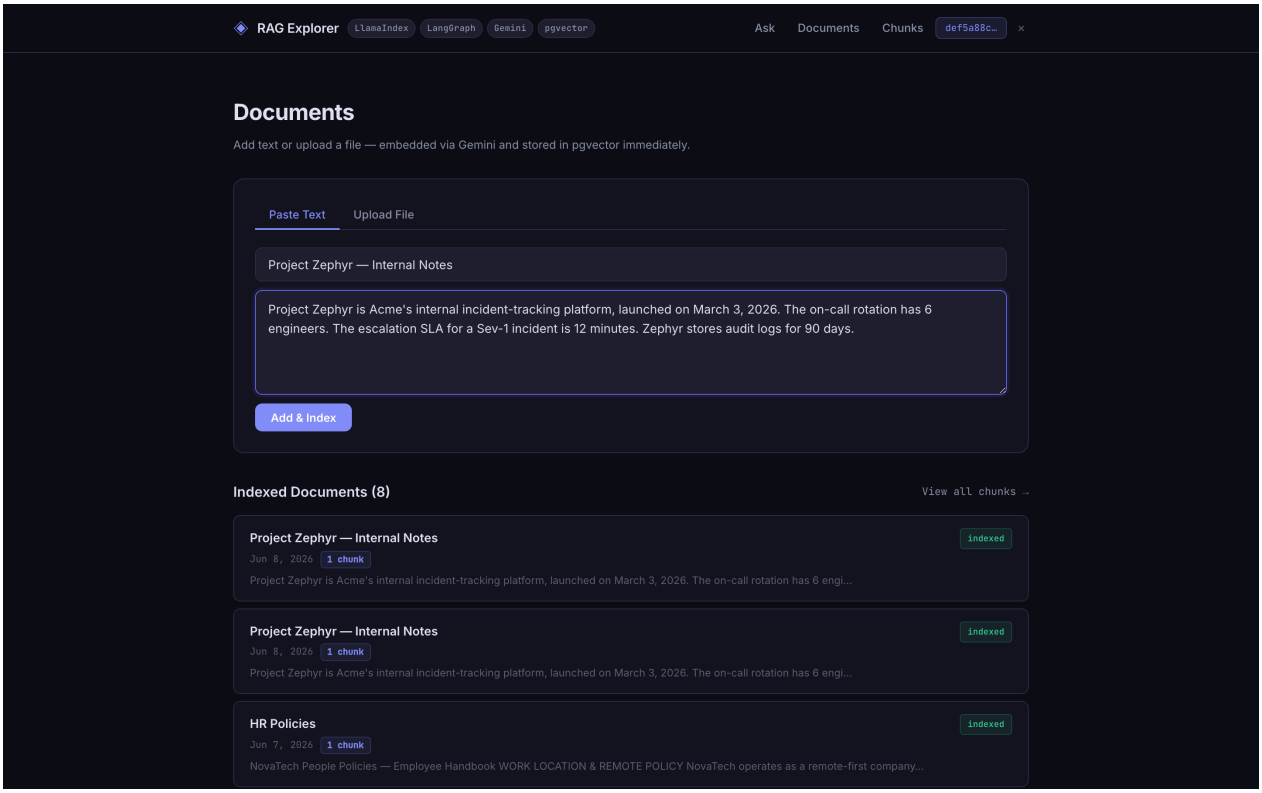


Figure 3 Documents page — the user pastes a custom document ("Project Zephyr — Internal Notes"). On `Add & Index`, Django chunks the text, embeds it with Gemini, and stores the vectors in pgvector. The document appears in the indexed list instantly.

This is the core value loop: the user supplies their *own* content — not seed data. Text paste or `.txt/.pdf/.md` upload both flow through the same LlamaIndex ingestion path and become queryable within seconds.

4.4 Ask About Your New Knowledge

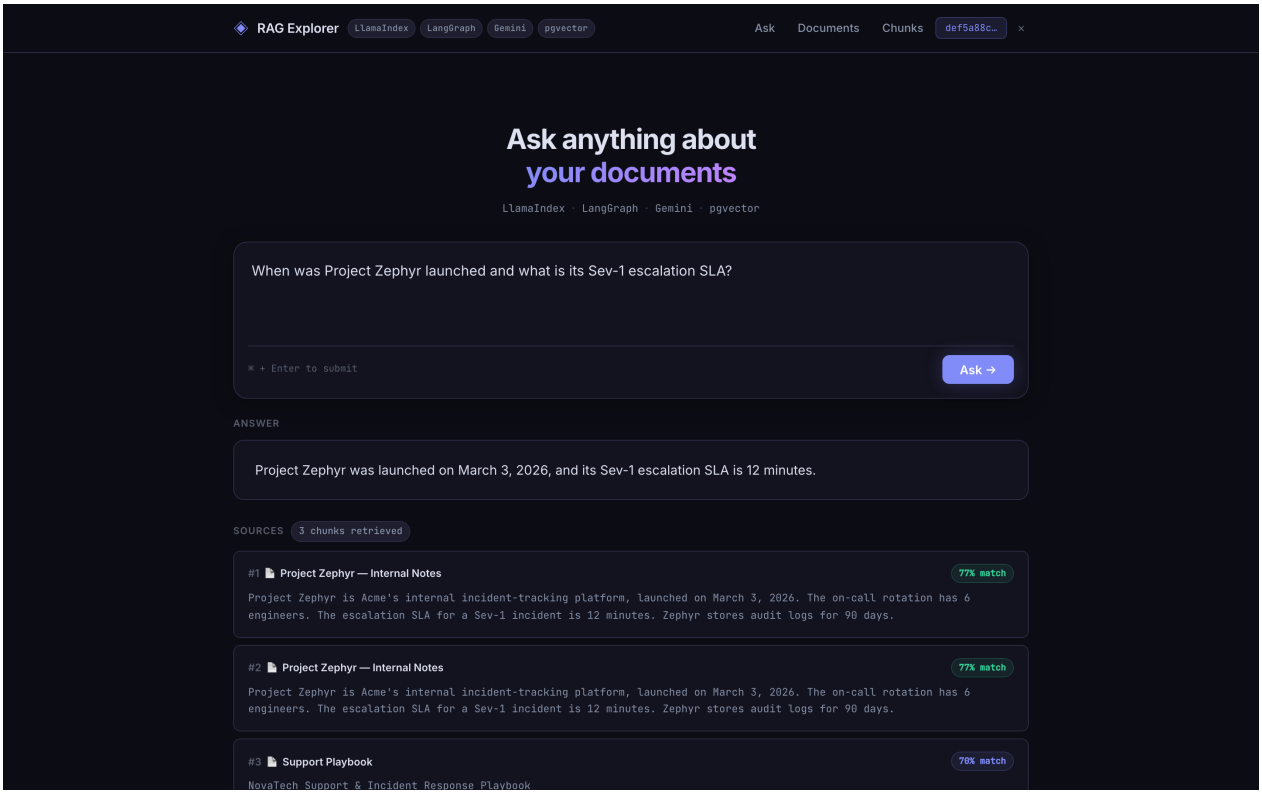


Figure 4 Custom query against the just-added document — ”When was Project Zephyr launched and what is its Sev-1 escalation SLA?” The grounded answer (launched March 3 2026, 12-minute SLA) cites the exact document the user added moments earlier, with relevance scores (77% match).

The query runs through the LangGraph pipeline (pgvector retrieve → Gemini generate) with a live animation. Critically, the top source card is the *user's own document* — proving the answer is grounded in the freshly-indexed content, not the seed corpus. Every answer is fully auditable.

4.5 Vector Chunk Explorer

The Chunks page exposes the raw pgvector content — the exact passages the retriever searches at query time. A live filter lets users inspect which chunks exist for a given document, making the system fully transparent and debuggable.

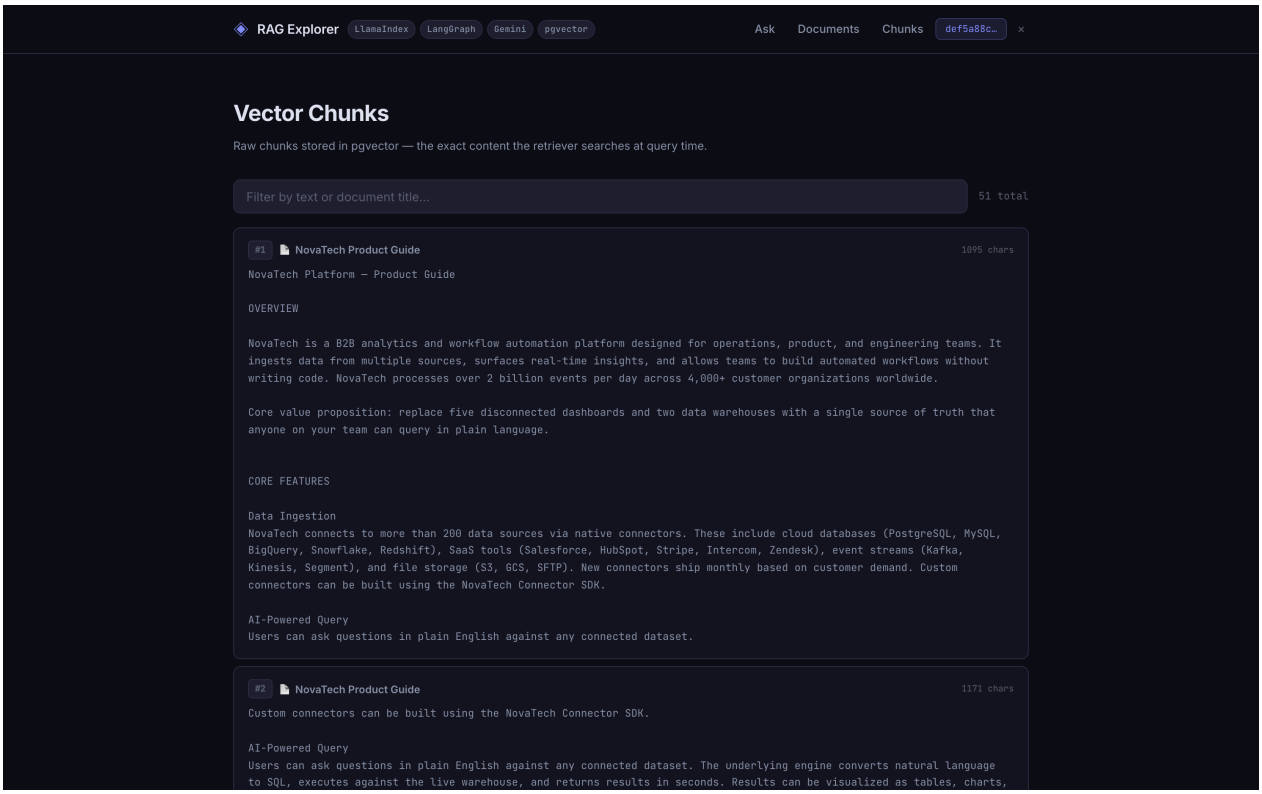


Figure 5 Chunks view — the pgvector chunks behind the corpus, with live search filtering. Each chunk shows its source document and character count.

5 Tech Stack

Layer	Technology	Role
Embedding & Retrieval	LlamaIndex + Gemini Embedding 2	Chunk, embed, retrieve (3072-dim vectors)
Orchestration	LangGraph	Retrieve → Generate state machine
LLM	Gemini (gemini-3.5-flash)	Answer generation from retrieved context
Vector Store	pgvector (PostgreSQL 16)	ANN search over document embeddings
Backend API	Django REST Framework	Token auth, document ingestion, query endpoint
WSGI	Gunicorn (4 workers)	Production app server
Frontend	Next.js 15 (static export)	React UI, deployed to Cloudflare Pages
Edge Proxy	Cloudflare Pages Functions	API proxy — same-origin, no CORS issues
Reverse Proxy	Nginx Proxy Manager	Routes subdomain to backend container
Infrastructure	Proxmox VM + Docker Compose	Self-hosted backend, pgvector sidecar

5.1 Key Engineering Decisions

LangGraph over raw LlamaIndex pipeline: LangGraph's typed state machine makes it straightforward to add new nodes (re-ranker, query expansion, guardrails) without refactoring the retrieval logic.

Cloudflare Pages Function as API proxy: rather than exposing the backend directly or managing

CORS headers, the Pages Function proxies `/api/*` from the same origin — browsers never touch a cross-origin request.

Source title injection: early testing revealed the LLM said "context does not contain NovaTech information" even when relevant chunks were retrieved — because raw chunk text doesn't repeat the company name. Fixed by prepending `(Source: {title})` to each chunk in the context string.

Expiring tokens over OAuth: for a controlled demo tool, time-limited UUIDs are simpler and auditable. No signup flow, no email verification — just generate and share.

6 Deployment

- **Frontend**: rag.projects.serhatkeskin.com — Cloudflare Pages, global CDN, automatic HTTPS
- **Backend API**: `api.internal.example.com` — self-hosted Proxmox VM (private subnet), Docker Compose, Gunicorn
- **Source code**: github.com/serhatkeskin/rag-explorer

The stack is fully reproducible: `docker compose -f docker-compose.prod.yml up` brings up the backend and pgvector. The frontend is a static export deployed with `wrangler pages deploy`.

7 Contact

Serhat Keskin

github.com/serhatkeskin

rag.projects.serhatkeskin.com

Demo token available on request.