FileNest

Case study by Naveen Raj — Full Stack Developer

Python FastAPI SQLAlchemy PostgreSQL Redis NATS JetStream OpenSearch Next.js React BetterAuth TypeScript Docker

Foundation (auth, uploads, file CRUD), the processing & events pipeline (virus scanning, MIME validation, NATS, webhooks, multipart), and the console app are complete; SDKs are in progress, with metadata/search, production hardening, and compliance features planned next.

Problem

Every application that accepts file uploads ends up rebuilding the same infrastructure from scratch — resumable uploads, virus scanning, MIME validation, multi-tenant isolation, signed webhooks, search, and compliance controls (WORM, legal hold, retention).

Approach

I designed the system myself before any code was written — the service split (a BetterAuth IAM issuing OAuth 2.1/PKCE tokens, a FastAPI backend, a Next.js console), the outbox-pattern upload pipeline (upload ’ outbox event ’ NATS JetStream ’ async virus scan/MIME validation/classification ’ webhook delivery), the tenant-isolation model, and the webhook signing scheme. The implementation itself was AI-assisted — "vibe coded": I fed in the architecture, the data flows, and the technical specs, and an AI coding agent built the FastAPI backend, the Next.js console, and the Node/React/Next.js/Python SDKs against that design.

Result

Foundation (auth, uploads, file CRUD), the processing & events pipeline (virus scanning, MIME validation, NATS, webhooks, multipart), and the console app are complete; SDKs are in progress, with metadata/search, production hardening, and compliance features planned next.