Nexus: Technical Design Report

AI-Powered Support Ticketing System

Vansh Taneja
Computing Science, University of Alberta
Edmonton, AB, Canada
vtaneja1@ualberta.ca

Kannan Khosla
Software Engineer, Pongo
Edmonton, AB, Canada
Kannan@pongo.ca

***Abstract*—Nexus is an AI-powered customer support ticketing platform built by co-founders Vansh Taneja and Kannan Khosla. It combines OpenAI GPT-4o-mini for first-line responses with human agent handoff, JWT-based authentication, role-based access control, email integration (SMTP, SendGrid; AWS SES stubbed for boto3), and Supabase for persistence and storage. This report describes the system architecture, backend design (FastAPI), AI integration and safety measures, deployment approach, scalability and security considerations, current limitations, and lessons learned. Repository: https://github.com/Kannan-Khosla/Nexus.**

I. Executive Summary

Nexus provides a full-stack support platform: users create tickets (web or email), the system optionally generates AI replies with rate limiting and PII sanitization, and admins manage tickets, SLAs, email accounts, and routing rules. The stack is FastAPI (Python), React (Vite), Supabase (PostgreSQL + Storage), and OpenAI. The system has been deployed at a client site (Pongo) and remains in production there. Repository: https://github.com/Kannan-Khosla/Nexus. This report summarizes architecture, key engineering decisions, and what is implemented in the codebase.

II. Problem Statement

Customer support teams face high volume and repetitive queries. Manual triage is slow; fully automated chatbots often lack context or escalate poorly. Nexus addresses this by: (1) automating first-line responses with a context-aware AI model, (2) threading conversations and maintaining state per ticket, (3) handing off to human agents when a ticket is assigned or escalated, and (4) supporting multiple channels (web UI and email) and admin tooling (priorities, SLAs, tags, routing rules). The goal is to reduce time-to-first-response and agent workload while keeping quality and control.

III. System Architecture

The system follows a three-tier design: React SPA (Vite), FastAPI backend, and Supabase (PostgreSQL + Storage). Email is ingested via IMAP polling or webhooks; outbound email uses SMTP, SendGrid, or AWS SES (SES integration is stubbed for boto3 in production). AI replies are generated by OpenAI only when no human is assigned; responses are rate-limited per ticket and sanitized before storage.

High-level architecture. Frontend (React/Vite) → FastAPI → Supabase DB/Storage; FastAPI → OpenAI; Email polling → FastAPI/Supabase.

Fig. 1. System architecture: three-tier (React, FastAPI, Supabase) with OpenAI and email ingestion.

IV. Key Engineering Decisions

- **FastAPI.** Chosen for async support, automatic OpenAPI docs, Pydantic validation, and type hints. Fits rapid iteration and clear API contracts.
- **Supabase.** PostgreSQL plus Storage reduced operational burden for a small team; the app uses Supabase as the primary data store and for file attachments.
- **Single AI model (GPT-4o-mini).** Balances cost and quality for support-style replies; no multi-model routing in the current codebase.
- **Rate limiting on AI replies.** Per-ticket, configurable window and max replies per window (e.g., 2 per 60s) to control cost and abuse.
- **Output sanitization.** All AI text is sanitized for profanity, emails, phone numbers, and credit-card-like patterns before storage and response.
- **JWT + role in token.** Auth is JWT-based; role (e.g., admin) is in the token payload and enforced in dependencies; optional legacy `X-Admin-Token` for certain admin endpoints.

V. Backend Design

A. FastAPI and Structure

The backend is a single FastAPI application (`main.py`) with Pydantic models for requests/responses, global exception handlers (middleware), CORS enabled, and a lifespan task for email polling. Configuration is via Pydantic Settings (`config.py`) with env validation. API failures: unhandled exceptions are caught by a global `error_handler` and return HTTP 500 with a generic message; validation errors return 422 with Pydantic `details`; HTTP exceptions (e.g., 403 for forbidden ticket access) are passed through `http_exception_handler`. No custom error codes or retry-after headers are defined.

B. API Endpoints (Representative)

Table I summarizes representative endpoints. The codebase exposes 60 routes: auth (register, login, forgot-password, reset-password, /auth/me); tickets (create, reply, get thread, rate, escalate, priority, time-entry, attachments, tags, category, send-email, emails); stats; admin (tickets, assigned, assign, close, delete, restore, trash, SLAs, email-accounts, email-templates, routing-rules, tags, categories); customer tickets; webhook (/webhooks/email). All ticket and attachment access is gated by get_current_user and role/ticket-ownership checks.

C. RBAC

Access control is role-based. get_current_user decodes the JWT and requires a valid token; get_current_admin enforces that the token role is admin. Ticket access: customers may only access tickets where user_id matches their id; admins can access all tickets. Admin-only endpoints use Depends(get_current_admin). Optional require_admin uses a shared X-Admin-Token header when configured. The application checks user_role (e.g., customer vs admin) for ticket visibility; customers may access only tickets where user_id matches.

D. Database Schema

The schema is defined by sequential SQL migrations (000–016). Core entities: users (id, email, password_hash, role, name); tickets (id, subject, status, user_id, assigned_to, priority, sla_id, source, is_deleted, etc.); messages (ticket_id, sender, message, confidence, success); attachments (ticket_id, message_id, file_path, etc.). Other tables: email_accounts, email_messages, email_templates, sla_definitions, routing_rules, routing_logs, tags, ticket_tags, categories, ratings, human_escalations, time_entries. File blobs live in Supabase Storage with DB attachment records.

Core entities. users → tickets → messages; email_accounts, email_messages; sla_definitions; routing_rules; ticket_tags, categories; attachments (Storage).

Fig. 2. Database schema (core entities and relationships).

VI. AI Integration

A. Model and Usage

The application uses OpenAI's gpt-4o-mini via client.chat.completions.create with a single user-message prompt. The prompt includes conversation history (concatenated sender/message) and instructions to reply as the support assistant. No tool use or function calling is implemented.

B. Safety and Guardrails

- **Sanitization.** sanitize_output() (main.py) applies regex-based redaction for profanity (PROFANITY), email (EMAIL), credit-card-like patterns (CC), and phone (PHONE). Redacted text and flags are returned; the redacted string is stored and sent to the client.
- **Rate limiting.** is_rate_limited(ticket_id) counts messages with sender == "ai" in the last ai_reply_window_seconds (default 60); if count ≥ ai_reply_max_per_window (default 2), the endpoint returns rate_limited: true and wait_seconds instead of calling OpenAI.
- **No AI when human assigned.** If ticket.assigned_to is set, the reply endpoint does not call OpenAI and returns a message that a human agent will respond.

C. Retries and Failure Handling

generate_ai_reply() (main.py) implements exponential backoff using openai_max_retries (default 3), openai_initial_delay (default 0.5s), and openai_backoff_multiplier (default 2.0). On failure after all attempts, the exception is re-raised; the ticket/reply endpoint does not catch it, so the global error_handler returns HTTP 500 with a generic message. No fallback reply or circuit breaker is implemented.

VII. DevOps and Deployment

A. Supabase

Supabase provides the PostgreSQL database and optional Storage. The app uses the Supabase client (anon key) and, for storage, a service-role key when configured. Migrations are run manually in the Supabase SQL editor in order.

B. Email (AWS SES and Others)

Outbound email supports SMTP, SendGrid, and AWS SES. The SES path in code is a stub that returns an error indicating boto3 should be used in production; SMTP and SendGrid are implemented. Inbound email is via IMAP polling (background task) or webhook (/webhooks/email).

C. Docker and CI/CD

The repository does not contain Dockerfiles or CI/CD configuration. Deployment is manual (e.g., uvicorn main:app and serving the frontend build). For client deployment (Pongo), environment variables and Supabase project are configured per environment.

D. Demo

The system is in production at a client site (Pongo). A short demo (create ticket, AI reply, admin assign/close) can be shared on request.

TABLE I
API ENDPOINT SUMMARY (REPRESENTATIVE).

Method	Path	Purpose
GET	`/`	Health
POST	`/auth/register`, `/auth/login`	Auth
GET	`/auth/me`	Current user
POST	`/ticket`	Create/continue ticket, optional AI reply
POST	`/ticket/{id}/reply`	Reply, optional AI
GET	`/ticket/{id}`	Thread
POST	`/ticket/{id}/rate`, `/escalate`	Rate, escalate
GET	`/admin/tickets`	List tickets (admin)
POST	`/admin/ticket/{id}/assign-admin`, `/close`	Assign, close
POST	`/admin/email-accounts`, `/email-templates`	Email config
POST	`/admin/routing-rules`, `/slas`, `/tags`	Rules, SLAs, tags
POST	`/webhooks/email`	Inbound email webhook

VIII. Scalability and Reliability Strategy

- **Stateless API.** FastAPI workers can be scaled horizontally behind a load balancer; session state is in JWT and DB.
- **Database.** Supabase/PostgreSQL can be scaled via plan upgrades and read replicas. Migrations define indexes on key columns (e.g., `tickets` user_id, priority, sla_id, is_deleted, assigned_to; `attachments` ticket_id; `email_messages` ticket_id).
- **AI and rate limits.** Per-ticket rate limiting caps OpenAI calls per conversation; at 10k users, global rate limiting and queueing would be needed to avoid OpenAI throttling and cost spikes.
- **Email polling.** Single process runs one polling loop; at scale, polling could be moved to a dedicated worker or replaced with webhook-only ingestion.
- **Reliability.** No health checks beyond `GET /`; no circuit breaker on OpenAI; failed AI calls surface as 500. Retries reduce transient failures but do not guarantee delivery.

IX. Security and Privacy Considerations

Passwords are hashed with bcrypt (`auth.py`). JWT secret and expiry are configurable (`config.py`); admin token is optional. CORS is set to `allow_origins=["*"]` in `main.py`. PII in AI output is redacted by `sanitize_output()` before storage. In `email_service.py`, `decrypt_credentials()` currently returns the value as-is (encryption not implemented). Scripts reference RLS (e.g., `disable_rls`); enforcement depends on Supabase settings. File uploads are validated by `MAX_FILE_SIZE` (10MB) and `ALLOWED_MIME_TYPES` in `storage.py`; attachment access is gated by ticket ownership/admin checks in the API.

X. Current Limitations

- No Docker/CI in repo; deployment is manual.
- AWS SES sending is not implemented (boto3 required).
- Email credential encryption is placeholder only.
- Single global error message on API failure; no structured error codes for clients.
- No circuit breaker or fallback when OpenAI is unavailable.
- CORS is permissive.
- Scale and load testing not documented; 10k users would require rate and capacity planning.

XI. Roadmap

Potential next steps (not committed in code): (1) Add Dockerfile(s) and CI (e.g., GitHub Actions) for build and test; (2) Implement SES sending with boto3; (3) Harden credential storage and restrict CORS; (4) Add health and readiness endpoints and optional circuit breaker for OpenAI; (5) Introduce global rate limiting and queueing for AI; (6) Document and run load tests for target scale.

XII. Lessons Learned

Building Nexus highlighted the importance of clear API contracts (FastAPI/OpenAPI), centralizing config (Pydantic Settings + env), and handling failure paths early (retries, sanitization, rate limits). Balancing feature speed with security (auth, PII, CORS) and operability (logging, migrations, deployment) is ongoing. The shared codebase uses an incremental migration strategy (e.g., role simplification in migration 014) while preserving existing behavior.

References

[1] Nexus repository. https://github.com/Kannan-Khosla/Nexus.

FastAPI. https://fastapi.tiangolo.com/.

Supabase. https://supabase.com/docs.

OpenAI API. https://platform.openai.com/docs.