Email Management and Auto-Response System

Overview:

This project combines a FastAPI backend with a React (Vite) frontend.
The backend handles ingestion, storage, NLP enrichment, and AI-based draft generation, while the frontend provides a real-time UI for managing emails and analytics.

The system is designed for flexibility:

- Works with demo data or live Gmail via IMAP.
- Supports multiple AI providers (Gemini, DeepSeek via OpenRouter, with local fallback).
- Includes real-time updates using Server-Sent Events (SSE).

High-Level Architecture:

- **Backend:** FastAPI REST + SSE server
- **Frontend:** React SPA (Vite dev server)
- **Database:** SQLite (with lightweight migrations)
- **Data Flow:**
  Provider (Demo/Gmail) → Ingestion Layer → SQLite DB → NLP (Sentiment/Priority) → AI Draft Generation → UI
- **Background Services:**
  - Email fetcher (thread-based)
  - Queue worker for deferred AI responses
- **AI Layer:** Pluggable LLM abstraction (Gemini, DeepSeek, local fallback, chain fallback).
- **RAG (Retrieval-Augmented Generation):** Optional in-process vector store (if ML deps installed).
- **Real-time Updates:** SSE stream for status changes and keepalive messages.

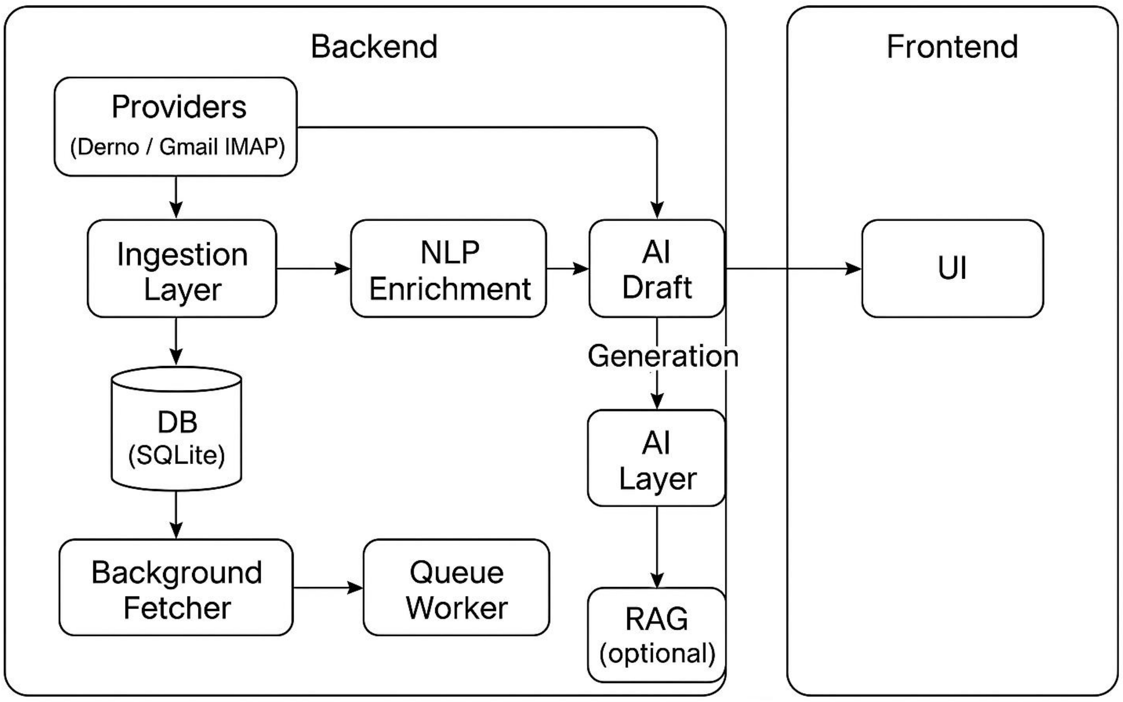

Figure 1: Architecture Diagram

<u>**Backend (FastAPI):**</u>

- **Framework:** FastAPI with async/sync blend.
- **Key Features:**
  - Initializes DB schema & migrations on startup.
  - Runs background fetcher & queue worker.
  - Supports switching between demo and Gmail providers.
- **Routers:**
  - /api/emails: CRUD, regenerate AI reply, approve, resolve, send simulation, fetch modes, maintenance tools.
  - /api/analytics: Sentiment & priority stats.
  - /api/events: SSE stream for live updates.
  - /health: Provider, email count, RAG status.
- **Security:** API Key via X-API-Key header (can bypass locally).

Data & Persistence:

- **Database:** SQLite (via SQLAlchemy ORM).
- **Email Model Fields:**
  id, sender, subject, body, received_at, sentiment, priority, auto_response, status, approved_at, sent_at, source, external_id
- **Migrations:** Lightweight (adds missing columns without Alembic).
- **Query Features:**
  - Filter by status/priority/sentiment/domain.
  - Fuzzy search & keyword filtering.
  - Hide demo data when Gmail mode is active.

Email Ingestion:

- **Providers:**
  - Demo (static/dataset).
  - Gmail (via IMAP, UID-based).
- **Deduplication:**
  - Uses Gmail UID (external_id).
  - Fallback dedupe: (sender, subject, timestamp).
- **HTML Handling:** Prefers text/plain, strips & sanitizes HTML when needed.

Background Systems:

- **Fetcher Thread:**
  - Polls Gmail/Providers at interval (EMAIL_POLL_INTERVAL).
  - Persists new emails → triggers AI draft generation.
- **Queue Worker:**
  - Handles deferred auto-responses (urgent prioritized).
- **Keepalive:** Sends SSE heartbeat every 15s.

NLP & Extraction:

- **Library:** textblob (sentiment analysis).
- **Priority Detection:** Heuristic keywords (e.g., "urgent").
- **Extraction:** Phone numbers, alternate emails, action keywords.
- **Maintenance Tools:** Recompute sentiment/priority across DB.

AI Layer (LLM Integration):

- **Providers Supported:**
  - Gemini (Google Generative AI)
  - DeepSeek (via OpenRouter)
  - Local static fallback
- **Config:** LLM_PROVIDER=gemini|deepseek|openrouter|fallback
- **Error Handling:**
  - Timeouts → fallback response
  - Quota errors → backoff
  - DeepSeek 402 credit errors → truncate & retry
  - Chain fallback (DeepSeek → Gemini → Local)
- **Diagnostics:**
  - /api/emails/ai/diag (provider status, last error)
  - /api/emails/ai/test (light prompt check)

RAG (Retrieval-Augmented Generation):

- **Optional**: Only enabled if sentence-transformers & faiss-cpu installed.
- **Modes:** off | lazy | sync
- **Knowledge Base:** Can add docs via /api/emails/kb/docs.

Frontend (React + Vite):

- **State Management:** React Query + Axios.
- **Realtime Updates:** SSE subscription to /api/events.
- **Components:**
  - **EmailList:** Paginated, filterable, searchable.
  - **EmailDetail:** Shows body, AI draft, approve/resolve actions.
  - **AnalyticsPanel:**
    - Sentiment bar (Chart.js)
    - Priority donut (ECharts)
    - Collapsible dashboard.
- **UI/UX Features:**
  - Debounced search (350ms).
  - Mode switching (Demo/Gmail).
  - Dataset recompute triggers.
  - Skeleton loaders.

Error Handling & Resilience:

- **Gmail Issues:** Safe returns on missing creds/network errors.
- **AI Issues:** Sentinel tokens ([GEMINI_UNAVAILABLE], [DEEPSEEK_ERROR]).
- **DB Issues:** Duplicate checks before insert.
- **Logging:** Errors captured with minimal PII.

Security:

- **Auth:** API Key header (simple, dev-focused).
- **Safe Rendering:** No raw HTML in frontend (prevents XSS).
- **.env Management:** Secrets not logged or committed.

Performance & Scaling:

- **Prototype:** SQLite + in-process workers.
- **Scaling Path:**
    - Switch to Postgres.
    - Use Celery + Redis for background workers.
    - Externalize RAG to Pinecone/Weaviate.
    - WebSockets or pub/sub for larger-scale real-time.

Running the Project:

Prerequisites:

- Python 3.9+
- Node.js + npm
- Install backend dependencies:
- pip install -r requirements.txt
- Install frontend dependencies:
- cd frontend
- npm install

Start Backend:

```
.venv\Scripts\Activate.ps1
uvicorn backend.app.main:app --host 127.0.0.1 --port 8000 --reload
```

Start Frontend:

```
cd frontend
npm run dev
```

Access frontend at: http://localhost:5173
Backend API at: http://127.0.0.1:8000

Conclusion:

This system helps manage emails in real time with AI support. It connects to different providers, adds basic NLP insights, creates draft replies, and shows useful analytics. Right now, it’s great for local or dev use, but it’s also built so it can easily grow and scale for production later.