System Overview

Last Updated: August 22, 2025

This guide explains the moving parts you'll use to build the chatbot on the sunshinespend.com intern server.

Components

- **Team site**: `https://teamX.sunshinespend.com/` (protected by Basic Auth)
- **Flask API**: `https://teamX.sunshinespend.com/pyapp`
  Entrypoint: `POST /bot` → proxied to `POST /pyapp/ask`
- **OpenAI proxy**: `http://127.0.0.1:8088` (holds the **shared ChatGPT API key**)
- **MySQL**: per-team database (`team0Xdb`) for physician data, etc.
- **PostgreSQL (Vector):** per-team database (teamX) for vector data (RAG support)
- **Optional remote APIs**: e.g., CMS Open Payments, NPI registry, or internal endpoints.

Request flow

1. Client calls `POST https://teamX.sunshinespend.com/bot` with JSON `{ "question": "..." }`.
2. Apache proxies to your team's Flask `/pyapp/ask` handler.
3. Your code can:
   - query **MySQL** (local) for physicians,
   - call **remote APIs** for fresh data,
   - ask **ChatGPT** via the local proxy (no API keys in your app).
4. You return a concise, safe answer to the client.

Files you edit

- `/srv/groups/team0X/python/wsgi.py` → your Flask app (already scaffolded)
- Helpers:
  - `chatgpt_client.py` (calls the OpenAI proxy; **no subkeys needed**)
  - `db.py` (parameterized MySQL queries)
  - `remote_api.py` (HTTP GET example you can adapt)

Guardrails & tips

- Keep answers **concise**; state uncertainty.
- Sanitize and **parameterize** all SQL.
- Log minimal PII; follow least-privilege principles.
- Prefer stable fields (NPI) for physician identity.
- Cache remote calls when appropriate.