Task 1: Security & Governance Gatekeeper

Owner: Backend Engineer

Objective

Add a backend security layer that redacts PII from AI responses, enforces role-based limits, and logs all activity for compliance.

Requirements

- **PII Redaction**
  Scan AI outputs for personal data (e.g. names, emails, phone numbers).
  If the user is not a privileged role, mask detected values using labels like `[REDACTED_EMAIL]`.

- **Sanitization Hook**
  Apply redaction in a centralized response-processing step so all AI outputs are consistently protected.

- **Audit Logs & Dashboard**
  Store request metadata (user, role, department, usage, PII flags).
  Provide an internal admin view to inspect logs and usage by department.

- **Role-Based Rate Limiting**
  Enforce request quotas per user role and record limit violations.

Key Components:

- **PII Redaction (The Filter):** Using a tool like **Microsoft Presidio**, the system scans text for patterns (regex) and entities (NLP) like social security numbers or emails. If the user isn't an "Admin," the backend swaps "john.doe@email.com" for `[REDACTED_EMAIL]` before the user ever sees it.
- **Sanitization Hook:** This is a centralized function in your Flask app. Instead of writing redaction code for every single chat endpoint, you route all AI responses through this one "hook" to guarantee consistency.
- **Audit & Governance:** You're building a trail of breadcrumbs. Every time the AI is used, you log who used it, their department, and if any PII was caught. This is critical for industries like Finance or Healthcare.
- **Rate Limiting:** Using **Redis**, you track how many requests a user makes. If a "Junior Intern" role is limited to 10 requests an hour, Redis counts them and blocks the 11th.

Success Criteria

- PII is never exposed to non-privileged users.
- Admins can monitor usage and limits per department.

Tech Stack is open to any personal preference or similar to :

- **Backend:** Flask, PostgreSQL, Redis, Microsoft Presidio (PII).