PRD: Internal RAG Retrieval + Citation Grounding Platform

1. Overview

This product delivers an internal Retrieval-Augmented Generation (RAG) platform for operators and engineers.
It provides grounded answers from trusted internal sources (`runbooks`, `incident timelines`, and `configs`) with explicit citations to improve auditability and decision confidence.

2. Problem Statement

Operational responders lose time searching scattered documentation during incidents and postmortems.
Without source-grounded answers, trust in AI responses is low and verification is manual and slow.

3. Goals

- Provide fast, relevant retrieval over internal ops documents.
- Ground every generated answer with explicit source citations.
- Support hybrid retrieval quality (semantic + lexical).
- Deliver a working operator UI with real-time responses.
- Enable reliable backend and test coverage for iterative shipping.

4. Non-Goals

- Replacing incident management tooling.
- Building a generalized public search engine.
- Performing write actions in external systems from AI responses.

5. Users

- SRE / Platform Engineers
- Incident Commanders
- On-call Responders
- Internal support engineers

6. Scope and Functional Requirements

6.1 Data Sources

- Runbooks (`data/runbooks`)
- Incident timelines (`data/incidents`)
- Configuration and operational docs (`data/configs`)

6.2 Ingestion + Indexing

- Parse markdown, text, PDF, and common config formats.
- Chunk documents with configurable chunk size/overlap policy.
- Extract useful metadata (`source`, `filename`, `type`, `section_title`, `chunk_index`, optional URLs).
- Generate embeddings for all chunks.

- Index vectors into pluggable stores:
  - Pinecone (managed)
  - FAISS (local)
  - Chroma (local/embedded)

6.3 Retrieval

- Top-K retrieval endpoint and internal retriever service.
- Hybrid retrieval:
  - Embedding similarity (dense retrieval)
  - BM25-style lexical scoring
- Optional reranking hooks to improve top result precision.

6.4 Grounded Generation

- Build prompt context from retrieved passages with source labeling.
- Produce structured citations in API response:
  - `citation_id`, `source`, `filename`, `doc_type`, `section_title`, `relevance_score`, `passage`, optional `url`.
- Normalize citation references in generated text to source labels.

6.5 API + UI

- FastAPI backend for:
  - health/status
  - retrieval/query
  - ingest
  - model listing/switch
  - benchmark comparison
- Chat-style frontend:
  - model select
  - plugin toggles
  - streaming response support
  - citations panel with source highlighting

7. Quality Requirements

- Answers must include citations when source evidence is available.
- Endpoint latency should remain operationally useful for on-call workflows.
- Graceful fallback to local/mock model when external keys are unavailable.
- Maintain deterministic unit tests through proper singleton/factory mock targeting.

8. Security and Compliance

- Internal-only data access and deployment.
- No sensitive secrets embedded in tests or frontend assets.
- Citation trail preserved for post-incident auditing.

9. Success Metrics

- Query success rate with citations.
- Median and p95 API latency for `/api/chat` and retrieval flows.
- Time-to-answer reduction in incident workflows.
- Operator trust signals (adoption and repeat usage).

10. Delivery Plan

Phase 1

- Stable ingestion + retrieval + grounded answer generation.
- UI that exposes source citations and model metadata.
- Baseline tests and smoke checks.

Phase 2

- Expanded test matrix and performance/regression gates.
- Additional connectors and model routing enhancements.
- Operational dashboards and reliability hardening.

11. Demo Acceptance Checklist

- [x] Backend starts and serves UI.
- [x] `/health` endpoint returns healthy status.
- [x] `/api/models` and `/api/plugins` respond correctly.
- [x] `/api/chat` returns an answer and citation payload contract.
- [x] Test suite passes with clean targeted API/CLI/ingestion tests.