Product Requirements Document (PRD): Pharmaceutical RAG Agent

1. Project Overview

The goal is to build a cost-effective Retrieval-Augmented Generation (RAG) agent that can accurately answer complex pharmaceutical questions by retrieving information from both unstructured text and structured tables.

- **Problem Statement**: Pharmaceutical data is fragmented across text and tables. Standard RAG systems often "break" table integrity during vectorization, leading to factual errors. High API costs for powerful LLMs also make large-scale deployment unfeasible.

- **Proposed Solution**: A hybrid system that stores text in a **Vector Database** and tables in a **SQL Database**. A two-tier **Model Cascade** routes simple classification tasks to a cheap local LLM and complex synthesis to a paid API.

2. Target Audience

- **Pharmaceutical Researchers**: Need precise, evidence-backed answers from research papers (PMC-OA) and drug labels (DailyMed).

- **Data Scientists/Admins**: Responsible for monitoring system performance and cost-efficiency (CER).

3. Core Features (MoSCoW Prioritization)

Must Haves (High Priority)

- **Hybrid Ingestion Pipeline (ETL)**: Extract text and tables from XML/PDF sources; load text into Vector DB and tables into SQL DB.

- **Tier-1 Query Classifier**: A local LLM (e.g., Llama 3 via Ollama) to categorize queries as "text-only," "SQL-only," or "hybrid".

- **Dual Retrieval Engines**:
    - **Vector Retrieval**: Semantic search for unstructured text.
    - **SQL Retrieval**: Precise factual lookups for structured data.

- **Tier-2 Answer Synthesis**: A paid API (e.g., GPT-4) to synthesize context from both sources into a final answer.

Should Haves

- **Source Referencing**: Provide citations for every retrieved fact to ensure traceability.
- **CER Monitoring**: Real-time tracking of API costs and latency to calculate the Cost-Efficiency Ratio.

Could/Won't Haves

- **Web UI**: A basic interface for submitting natural language queries.
- **Multi-user Support**: Not required for this prototype.

4. Technical Architecture

- **Backend**: Python with **FastAPI**.
- **Orchestration**: **LangChain** for RAG logic and LLM routing.

- **Storage**:
  - **Vector DB**: PGVector or ChromaDB.
  - **SQL DB**: PostgreSQL or SQLite.
- **Models**:
  - **Local (Tier-1)**: Llama-3-8B or Mistral (via Ollama).
  - **Remote (Tier-2)**: GPT-4o or Claude 3.5 Sonnet.

---

5. Cursor IDE Implementation Plan

To build this efficiently in Cursor, follow these steps to leverage its AI-native features:

Step 1: Initialize Project and `.cursorrules`

Create a `.cursorrules` file in your root directory to provide the AI with context about your stack and coding standards.

Suggested `.cursorrules` content:

> "You are an expert Python developer building a Hybrid RAG system. - Follow Object-Oriented Programming (OOP) principles.

- Use FastAPI for the API layer and LangChain for orchestration.
- Always include source references in LLM responses. - Prioritize modularity for the IngestionWorker, VectorClient, and SQLClient components."

Step 2: Use Composer for Component Development

Use **Cursor Composer (Cmd+I)** to generate the core modules.

- **Prompt for Ingestion**: "Build a modular Python pipeline that takes PMC-OA XML files, extracts text sections for a Vector DB and tables for a SQL DB using the 'DailyMed' structure as a reference."
- **Prompt for Routing**: "Create a Tier-1 Query Classifier using LangChain and a local Ollama model. It should output a JSON classification: {type: 'text' | 'sql' | 'hybrid'}." +1

Step 3: Codebase Indexing

Ensure Cursor indexes your entire project. This allows you to ask cross-file questions like:

- *"How does the `ModelCascadeRouter` pass retrieved SQL rows to the Tier-2 synthesis prompt?"*
- "Check if the `IngestionWorker` is correctly preserving table headers during SQL insertion."

Step 4: Testing & Iteration

Use the **Terminal** within Cursor to run independent tests for the Tier-1 classifier.

- **Command**: `pytest tests/test_classifier.py`
- **Refinement**: If classification accuracy is low, highlight the prompt code and ask Cursor: "Refine this prompt to better distinguish between a 'hybrid' query and a 'text-only' query for pharmaceutical data."

6. Success Metrics (KPIs)

- **Query Classification Accuracy**: >90% F1-score for Tier-1 routing.
- **Latency**: Full hybrid synthesis in <15 seconds.
- **CER (Cost-Efficiency Ratio)**: Must be significantly lower than a "GPT-4-only" baseline.