Technical Design Document

Project: AI Empower – Enterprise RAG Service

Version: 2.0 (Current State) & 3.0 (Roadmap) **Date:** November 21, 2025 **Author:** Greg Burns

1. Executive Summary

The objective of this project is to build a scalable, multi-tenant Retrieval-Augmented Generation (RAG**)** service on Google Cloud Platform (GCP). The system is designed to ingest unstructured data (PDFs, Medical Textbooks), index them for semantic search, and provide accurate, context-aware answers via a conversational API.

Version 2.0 (Current State) achieves an event-driven, serverless architecture using Cloud Functions (Gen2), Eventarc, and Firestore Vector Search. It prioritizes stability, handling transaction limits, and cold-start optimization.

Version 3.0 (Planned) will introduce multi-tenancy, real-time streaming performance, and self-service document management via the frontend.

The Streamlit front-end for this app is currently live at https://test-rag-frontend.streamlit.app/.

2. System Architecture (V2.0)

The system is divided into two decoupled microservices: **Ingestion** and **Retrieval**.

2.1 High-Level Flow

1. **Ingestion:** `PDF Upload` -> `GCS Bucket` -> `Eventarc` -> `Cloud Function` -> `Vertex AI Embedding` -> `Firestore`.
2. **Retrieval:** `User Request` -> `Cloud Function` -> `Vertex AI Embedding` -> `Firestore Vector Search` -> `Gemini 2.5 Pro` -> `Response`.

2.2 Component Stack

- **Compute:** Google Cloud Functions (2nd Gen).
- **Storage (Raw):** Google Cloud Storage (GCS).
- **Storage (Vector/Meta):** Firestore (Native Mode) with Vector Search.
- **AI Models:**
    - **Embedding:** `text-embedding-004` (768 dimensions).

    - **Generation:** `gemini-2.5-pro` (Chosen for superior reasoning on complex medical texts).
- **Orchestration:** Eventarc (GCS Triggers).
- **Frameworks:** LangChain, Python 3.11.

3. Technical Specifications (V2.0)

3.1 Ingestion Service

Trigger: `google.cloud.storage.object.v1.finalized` **Endpoint:** Internal (Event-driven)

Workflow:

1. **Event Detection:** Eventarc detects a new file in `test-rag-backend-bucket`.
2. **Validation:** Function filters for `.pdf` extensions.
3. **Processing:**
    - File downloaded to ephemeral storage (`/tmp`).
    - `PyPDFLoader` extracts text.
    - `RecursiveCharacterTextSplitter` chunks text (Size: 1000 chars, Overlap: 100).
4. **Vectorization:** Vertex AI generates 768-dim vectors for each chunk.
5. **Persistence:**
    - Chunks are wrapped in `google.cloud.firestore_v1.vector.Vector` objects.
    - **Batch Management:** Writes are committed in batches of **50** to strictly adhere to Firestore's 10MB transaction limit.

3.2 Retrieval Service

Trigger: HTTP Request **Endpoint:** Public HTTPS (currently unauthenticated for POC)

Workflow:

1. **Initialization:** Uses **Strict Lazy Loading**. Heavy libraries (`langchain`, `vertexai`) are imported inside the handler to prevent container health-check timeouts.
2. **Embedding:** User query is converted to a vector using `text-embedding-004`.
3. **Search:** Executes `collection.find_nearest()` using **Cosine Distance**.
    - *Limit:* Top 5 results.
4. **Generation:**
    - Retrieved context is concatenated.
    - Gemini 2.5 Pro generates a response based *only* on the provided context.
5. **Response:** Returns JSON containing the answer and source metadata.

3.3 Data Schema (Firestore)

Collection: `rag_knowledge_base`

Field Name	Type	Description
`source`	String	Filename (e.g., "Surgery_Handbook.pdf")
`content`	String	The raw text chunk used for context.
`page`	Integer	Page number from source PDF.
`embedding`	Vector	768-dimensional float array (indexed).

4. V3.0 Roadmap & Enhancements

The following features are approved for the next iteration to transition from "Robust POC" to "Enterprise Product."

4.1 Multi-Tenancy & Security

- **Requirement:** Isolate data between different clients (e.g., "Medical Institute A" vs. "University B").
- **Implementation:**
    - **Schema Update:** Add `client_id` and `tier` fields to Firestore documents.
    - **Ingestion Update:** GCS paths will include client IDs: `gs://bucket/{client_id}/uploads/`.
    - **Retrieval Update:** Vector Search queries will apply a **Pre-filter** mask to ensure users only search their own `client_id`.

4.2 Performance Optimization

- **Warm Instances:** Configure Cloud Functions with `min_instances=1` to eliminate the 5-10s cold start latency.

- **Streaming Responses:** Update the API to return `Transfer-Encoding: chunked` (Server-Sent Events), allowing the UI to display the answer character-by-character as Gemini generates it.
- **Semantic Caching:** Implement a hash-map cache in Firestore. If a question is semantically identical to a recent query, return the cached answer instantly (< 200ms).

4.3 Advanced Accuracy

- **Hybrid Search:** Implement a dual-pass search merging **Vector Similarity** (Concept match) with **BM25/Keyword Search** (Exact term match). This is critical for specific medical drug names.
- **Chat Memory:** Persist conversation history in a separate Firestore collection (`chat_sessions`) to allow follow-up questions.

4.4 Frontend Enhancements (Self-Service)

- **Requirement:** Enable users to upload new PDF documents directly from the Streamlit Interface.
- **Implementation:**
    - **UI Component:** Integrate `st.file_uploader` into the Streamlit sidebar.
    - **Logic:** The Streamlit app will authenticate using the Service Account and upload the file directly to the `test-rag-backend-bucket` GCS bucket.
    - **Feedback Loop:** The UI will poll for ingestion status to notify the user when the document is ready for searching.

5. Deployment & CI/CD

- **Repository:** GitHub (`vertex-rag-firestore`).
- **Infrastructure as Code:** Currently scripted via `gcloud` CLI commands in Notebook.
- **Environment Variables:**
    - `GCP_PROJECT`: test-rag-backend.
    - `GCP_REGION`: `us-central1`.
    - `LOG_LEVEL`: `INFO`.

6. Known Limitations (V2.0)

- **Index Latency:** New documents take 5-15 minutes to become searchable due to Firestore index backfilling.
- **No Memory:** Each query is treated as independent; no conversation history is retained.
- **Transaction Limits:** Batch size strictly capped at 50 to avoid `400 Transaction Too Big`.