Daniel Adama Submission

RAG Retrieval System Architecture Components Documentation

1) Component Name: Search Endpoint

- Purpose
  - Receives the client request (HTTP) and forwards the validated search payload into the RAG system.
- Limitations
  - Bounded by API server throughput, auth checks and concurrent connections.
  - Can add latency to the end-to-end path if synchronous.
  - Vulnerable to malformed requests or injection if validation is weak.
- Necessity – Critical
  - It is the external integration point; without it clients cannot initiate queries.
- Alternatives
  - gRPC or WebSocket endpoints for lower-latency or streaming needs.
  - Message-queue fronting (async) for workloads that tolerate deferred responses.

2) Component Name: Search Payload

- Purpose
  - The structured data representing the client's request (query text, filters like user_id/date range, ref_time, k, etc.) that the API passes to the RAG manager.
- Limitations
  - Poorly designed payloads (missing fields, inconsistent date formats) break downstream steps.
  - Large payloads or heavy filter sets can complicate routing and validation.
- Necessity – Critical

- Defines what to search for and how to scope it; essential for correct retrieval.

- Alternatives
  - Split into smaller payloads (metadata only + separate query body) for efficiency.
  - Use a canonical request schema (e.g., JSON Schema) to standardize and version payloads.

3) Component Name: RAGManager

- Purpose
  - Facade/orchestrator that accepts the search payload, validates and translates it into internal search parameters, coordinates the retrieval pipeline, applies post-processing and returns formatted results.
- Limitations
  - Single point of orchestration, bugs can break full flow.
  - If it performs heavy synchronous computation (e.g., reranking) it can become a throughput bottleneck.
  - Needs robust error handling to avoid leaking upstream exceptions.
- Necessity – Critical
  - Central coordinator; standardizes how queries are processed, enforces policies and funnels results to clients.
- Alternatives
  - Split into smaller services (Gateway + Orchestrator) to separate concerns and scale independently.
  - Move some responsibilities (validation, auth, simple filtering) to an API gateway.

4) Component Name: Search Parameters

- Purpose
  - Internal representation of the payload after translation by RAGManager (e.g., limit, threshold, filters, ref_time) used to configure the retrieval pipeline.

- Limitations
  - Incorrect mapping or defaults can produce irrelevant or empty results.
  - Complex parameter combinations increase logic complexity and testing surface.
- Necessity – Important
  - Necessary to control retrieval behavior; less critical than RAGManager itself but required to make search configurable and reproducible.
- Alternatives
  - Use a policy/config service or feature flags to centrally manage default parameters.
  - Adopt typed configuration model or schema to prevent invalid combinations.

5) Component Name: Retrieval Pipeline

- Purpose
  - The staged flow that orchestrates query filtering, encoding, vector search and post-processing (logical layer that executes search parameters).
- Limitations
  - As features (hybrid retrieval, reranker) are added the pipeline grows complex and harder to maintain.
  - Resource contention if encoding and searching run on same limited nodes.
  - Synchronous pipeline steps can increase tail latency.
- Necessity - Critical/Important
  - Functionally essential; it can be refactored into microservices, but the pipeline concept must exist to perform retrieval.
- Alternatives
  - Implement pipeline as separate worker services chained by a task queue (more resilient & scalable).
  - Use an off-the-shelf retrieval pipeline framework if you want a managed orchestration layer.

6) Component Name: User Query

- Purpose
  - The raw textual query string (or search expression) that the user wants to retrieve documents for.
- Limitations
  - Ambiguity or noise in natural language can reduce retrieval precision.
  - Very long queries may need truncation or special handling.
- Necessity - Critical
  - The query is the core input; retrieval without it is meaningless.
- Alternatives
  - Structured queries (fielded search) for precise needs.
  - Query expansion / normalization step (spell-correct, entity extraction) before encoding.

7) Component Name: Query Encoder / Vectorizer (SentenceTransformer — same model used for indexing)

- Purpose
  - Transforms the user query into a dense vector embedding in the same vector space used during indexing so that similarity search is meaningful.
- Limitations
  - Throughput constrained by CPU/GPU; larger models need GPUs for low latency.
  - Model version drift or differing tokenizers between indexing and encoding breaks alignment.
  - Per-query latency vs batch throughput tradeoff.
- Necessity – Critical
  - Dense retrieval requires consistent embeddings for both queries and documents.
- Alternatives
  - Hosted embedding services (OpenAI/Cohere) to avoid local model ops.

    - Use a smaller/faster embedder (e.g., miniLM) for lower latency at some semantic cost.

8) Component Name: Encoded Query, Collection, and Filter

- Purpose
    - Packaged arguments sent to the vector store: the encoded query vector, the target collection name, and metadata filter/constraints that limit search scope.
- Limitations
    - Filters depend on consistent metadata; mismatches lead to no or incorrect results.
    - Large or complex filters can increase search time or cause unexpected behavior.
- Necessity – Important
    - Crucial for targeted searches and cost/precision control; the collection must be identified and filters applied to scope results.
- Alternatives
    - Pre-filter by querying metadata store to produce candidate IDs and then perform vector search only on that subset.
    - Keep filters minimal and apply more advanced filtering in a secondary DB after retrieval.

9) Component Name: Qdrant NN Search

- Purpose
    - Executes nearest-neighbor search in the vector database (Qdrant) using the query vector and filter; returns the closest vectors (candidates) by cosine distance.
- Limitations
    - Latency and memory usage scale with dataset size and index settings.
    - Exact search on large datasets is slow; ANN configurations trade recall for speed.
    - Quantization can reduce RAM but harm recall; requires tuning.
    - Dependence on Qdrant availability and cluster health.

- Necessity - Critical
    - This is the engine performing semantic similarity retrieval.
- Alternatives
- FAISS (self-hosted), Pinecone, Weaviate, Milvus, or Elastic with dense vectors depending on ops preferences and scale.

10) Component Name: Nearest Neighbors

- Purpose
    - The returned candidate set from Qdrant: nearest document vectors with associated payload metadata and similarity scores.
- Limitations
    - Candidate set can include semantically related but irrelevant hits (false positives).
    - Quantity and quality depend on k chosen and index configuration.
    - Score meanings depend on model and distance metric, requiring normalization/tuning.
- Necessity - Critical
    - Raw material for final selection/reranking; must be produced to continue.
- Alternatives
    - Combine with sparse keyword matches to produce a hybrid candidate set.
    - Pre-prune or cluster candidates to reduce noisy inputs.

11) Component Name: Results

- Purpose
    - An initial structured representation of hits (id, payload metadata, snippet/page_content, score) converted from Qdrant output for downstream filtering and formatting.
- Limitations
    - Returning full chunk text can increase payload size and network overhead.

    - Results may require sanitization (PII removal) before downstream use.
- Necessity - Critical
    - Necessary to present or further process the candidate documents.
- Alternatives
    - Return only IDs and metadata in the first phase and lazily fetch full text on demand.
    - Return summarized previews rather than full chunks to reduce transfer size.

12) Component Name: Search Threshold

- Purpose
    - A configurable gate that filters out results below a similarity score threshold to reduce noisy/low-quality hits.
- Limitations
    - Fixed thresholds can be brittle: too high → no results; too low → noisy results.
    - Score distributions change across datasets and embedders; threshold must be tuned and possibly adaptive.
- Necessity - Important
    - Improves precision and avoids handing irrelevant context to downstream LLMs, but must be tuned per collection.
- Alternatives
    - Dynamic thresholding (percentile-based) or use reranker score for final gating.
    - Return top-k unfiltered and let client decide (less safe for automated pipelines).

13) Component Name: Sorted Results

- Purpose

    - Final ordered list returned to the client after applying thresholding and any sorting (e.g., by score or temporal proximity); the canonical output of the retrieval flow.
- Limitations
    - Sorting strategy (score vs temporal proximity vs composite) must match user intent; wrong choice reduces usefulness.
    - If sorting mixes metrics (score + time), weights need careful tuning.
- Necessity - Critical
    - The consumer expects ordered, relevant results; sorting finalizes relevance.
- Alternatives
    - Provide multiple ranked views (by score, by time, by recency) so clients choose.
    - Expose additional metadata (score, time_distance) for client-side ranking.