# Phase 3 – Retrieval-Augmented Generation (RAG)

**🔹 RAG Architecture**

1. Ingestion Layer  
- Document upload, parsing, and chunking (PDF, DOCX, Markdown, SQL text).  
- Clean text storage + chunking by semantic boundaries.  
  
2. Embedding & Storage Layer  
- Generate vector embeddings (OpenAI or HuggingFace).  
- Store in Postgres + pgvector or Azure Cognitive Search.  
- Maintain metadata in SQL (docId, chunkId, tags, owner, version).  
  
3. Retrieval Layer  
- Semantic + hybrid search over embeddings.  
- Ranking, deduplication, filters.  
  
4. Augmentation + Generation Layer  
- Retrieved chunks + user query → LLM.  
- Compare baseline LLM vs RAG-enhanced responses.  
- Debug mode: show retrieved chunks in Admin Panel.

**🔄 Phase 3 Sub-Phases**

1. \*\*Ingestion Layer ✅ Completed (Phase 3.1)\*\*  
 - Document upload, parsing, and chunking (PDF, DOCX, Markdown, SQL text).  
 - Clean text storage + chunking by semantic boundaries.  
 - Stored documents + chunks in \*\*MS SQL\*\*.  
  
2. \*\*Embedding & Storage Layer ✅ Completed (Phase 3.2)\*\*  
 - Implemented \*\*OpenAI-based embedding generation\*\* (`text-embedding-ada-002`).  
 - Added \*\*Pgvector.EntityFrameworkCore\*\* support for `vector(1536)` type.  
 - Set up \*\*Postgres (Docker + pgvector)\*\* for vector storage.  
 - Created `EmbeddingService` to fetch chunks from SQL → generate embeddings → store in Postgres.  
 - Added \*\*Admin test UI\*\* (`/Embedding/Test`) → directly injected and called business services.  
  
3. \*\*Retrieval Layer ✅ Completed (Phase 3.3)\*\*  
 - Implemented `IRetrievalService` + `RetrievalService`.  
 - Supports Semantic Search (pgvector), Keyword Search (SQL Server FTS), Hybrid Search (merge + re-rank).  
 - Added \*\*Admin Debug UI\*\* with semantic vs hybrid comparison, filters, and tagging.  
  
4. \*\*Augmentation + Generation Layer ✅ Completed (Phase 3.4)\*\*  
 - Implemented `IRagService` + `RagService` for baseline vs RAG-enhanced answers.  
 - Integrated RetrievalService with LLM via augmented prompts.  
 - Added DTOs (`RagResultDto`, `RagChunkDto`) for structured outputs.  
 - Built Admin Debug UI (RAG Comparison Page) with query input, side-by-side answers, context view, latency metrics, and SLA badge.  
 - Logged results into SQL Server (`RagComparisonHistories`) for history tracking.

**⚡ Expected Outcomes**

- RAG-enabled chatbot in .NET + AdminLTE project.  
- Admin panel for document + RAG debugging.  
- Comparison: baseline vs RAG-enhanced responses.  
- Scalable + production-ready retrieval pipeline.

Architecture Diagram (Phase 3.4):

