SupplyChainGPT — RAG Retrieval Module (Focus Area)

This blueprint focuses on the **RAG layer**: ingesting, indexing, retrieving, and grounding the LLM's answers in internal business documents (SOPs, stock reports, contracts, emails/minutes, PDFs, Excel, CSVs).

---

1) Goals & Non-Goals

Goals - Ingest heterogeneous enterprise documents securely and continuously. - Normalize, chunk, embed, and index content with rich metadata. - Retrieve top-k relevant chunks with business-aware re-ranking. - Ground LLM responses with citations, provenance, and guardrails. - Support tenant/user/role-based access control (RBAC).

Non-Goals (for this workstream) - Building forecasting/optimization models (covered in ML layer). - Production infra hardening beyond minimal best practices.

---

2) High-Level Architecture

```
[Sources]
  ├─ SharePoint/Drive (PDF, DOCX, PPTX)
  ├─ S3/Blob (CSV, Parquet, JSON)
  ├─ Email/Minutes (EML/MSG/MD)
  └─ ERP/DB Exports (XLSX/CSV)
        │
        ▼
[Ingestion Workers]
  - File watchers + batch pulls
  - OCR (for scanned PDFs) + table extract
  - PII/Secrets redaction (policy-based)
        │
        ▼
[Document Normalization]
  - Convert to canonical text + structure
  - Split into semantic chunks
  - Attach metadata (source, doc_type, sku, warehouse, date_range, owner, acl)
        │
        ▼
[Embeddings + Index]
  - Text embedding model (bilingual if needed)
  - Vector DB (Chroma/FAISS/Pinecone)
```

```
  - BM25 sparse index for hybrid search
        |
        ▼
[Retriever]
  - Hybrid (dense + sparse) + re-ranking
  - ACL filter + time decay + business rules
        |
        ▼
[LLM Orchestrator]
  - System prompt + tools + citations
  - Answer synthesis + guardrails
        |
        ▼
[Client]
  - Chat UI + source previews + export
```

---

3) Data & Metadata Schema

Canonical chunk record (logical):

```
chunk_id: string (uuid)
doc_id: string (uuid)
source_uri: string (path or URL)
doc_title: string
chunk_text: string
chunk_tokens: int
embedding: vector<float>
doc_type: enum{policy, sop, contract, report, meeting_notes, email, manual,
export}
created_at: datetime
updated_at: datetime
effective_date_start: date|null
effective_date_end: date|null
sku_ids: array<string>|null
warehouse_ids: array<string>|null
supplier_ids: array<string>|null
region: string|null
version: string|int
language: string (e.g., en, hi)
acl: { tenants: [..], roles: [..], users: [..] }
provenance: { checksum, parser, ocr_confidence }
```

Indexing keys: `(doc_type, sku_ids, warehouse_ids, supplier_ids, effective_date_start/end)` used for pre-filters.

---

4) Ingestion & Normalization Pipeline

Connectors - Cloud drives (SharePoint/Google Drive/OneDrive): list + delta sync. - Blob storage (S3/Azure): prefix scans. - Email: IMAP pull for specific mailboxes/labels. - DB exports: fetch CSV/XLSX dropped to a landing zone.

Parsing - PDF: pdfminer + OCR fallback (Tesseract) with image-based page detection. - DOCX/PPTX: python-docx/python-pptx. - XLSX/CSV: pandas; convert key tables to markdown blocks. - Images (scanned docs): OCR + layout detection; store OCR confidence.

Chunking strategy - **Semantic chunking** by headings for policies/SOPs (H1/H2/H3). - **Table-aware** chunking for reports/exports (one logical table per chunk + short caption). - Token target ~ **600–800 tokens** per chunk (with 100–150 token overlap) to balance recall vs. precision.

Metadata enrichment - Heuristic/entity extraction for SKU, warehouse codes, supplier IDs, dates, regions. - Compute checksums for dedup/versioning. - Build ACL from folder path, file owner, or DLP labels.

---

5) Embeddings & Indexing

- **Embedding model**: sentence-transformers (e.g., `all-MiniLM-L6-v2`) for speed; consider domain-tuned model later.
- **Vector store**: start with **Chroma** (dev) → swap to Pinecone/Weaviate in prod.
- **Hybrid search**: Combine dense vectors with **BM25** (e.g., Elasticsearch/OpenSearch) for keyword/number matching.
- **Re-ranker**: Cross-encoder (e.g., `ms-marco-MiniLM-L-6-v2`) to re-rank top 100 by semantic relevance.
- **Freshness**: time decay in scoring; prefer latest version unless query requests historical.

Scoring idea: `score = α*dense + β*sparse + γ*re_rank + δ*freshness`.

---

6) Retrieval Policies & Business Logic

- **ACL filter first**: exclude chunks the user cannot access.
- **Pre-filters** from query intent: detect `SKU`, `warehouse`, `supplier`, `date_range` entities.
- **Document-type boosts** based on question: contracts for SLAs, SOPs for “how to”, reports for metrics.
- **Numeric intent**: when query seeks quantities/dates, boost chunks with tables/numbers.
- **Temporal routing**: if question concerns a past period, restrict to corresponding effective_date.

---

7) LLM Orchestration (Answering with Citations)

System prompt skeleton - Be concise, factual, and cite sources per passage. - Quote small excerpts (<=50 words) and include doc title + page/section if known. - If insufficient evidence, ask for a document or return “not enough data.”

Tools - `retriever.search(query, k, filters)` - `table_reader(read_chunk_id)` for accurate numeric extraction. - `policy_checker(checklist)` to validate recommendations against constraints (e.g., SLAs).

Answer template - Summary → Key points → Actionable recommendation → Citations list.

---

8) Guardrails, Redaction, Compliance

- **PII/Secrets redaction**: email, phone, card, API keys; redact before indexing.
- **Contractual clauses**: label risky clauses (penalties, exclusivity). Optionally block-list for chat exposure.
- **Hallucination control**: require $\geq$2 corroborating chunks for certain claim types; otherwise respond with uncertainty.
- **Safety**: profanity/offensive filter on outputs; log all prompts/answers with doc IDs.

---

9) Evaluation Plan (RAG-specific)

Offline - Build a gold set of ~150 Q&A pairs with ground-truth snippets. - Metrics: Retrieval **Recall@k**, MRR; Answer **Faithfulness**, **Factuality**, **Answer-contains-citation**.

Online - Implicit feedback: clicks on citations, time-to-first-doc, copy events. - Explicit feedback: thumbs up/down, “helpful?” scores, error tags (missing, outdated, wrong).

A/B compare: `dense`, `hybrid`, `hybrid+re-rank` retrievers.

---

10) Minimal Viable Implementation (MVP)

Tech: Python, FastAPI, Chroma, sentence-transformers, OpenAI/Llama, Streamlit.

Steps 1. One connector (e.g., a folder on disk) → parse PDF/DOCX/XLSX. 2. Chunk + embed with metadata; write to Chroma. 3. Build retriever (hybrid optional) with ACL filter (single tenant for MVP). 4. FastAPI endpoint `/ask`: runs retrieval → LLM synthesis with citations. 5. Streamlit UI: chat + source panel + file upload for ad-hoc docs.

---

11) Code Scaffolding (Illustrative)

Ingestion & Indexing (Python)

```
from pathlib import Path
import uuid, json, hashlib, pandas as pd
from sentence_transformers import SentenceTransformer
import chromadb
from chromadb.utils import embedding_functions

EMB_MODEL = "all-MiniLM-L6-v2"
model = SentenceTransformer(EMB_MODEL)
client = chromadb.Client()
coll = client.get_or_create_collection("supplychain_docs")

def checksum_bytes(b: bytes) -> str:
    return hashlib.sha256(b).hexdigest()

def chunk_text(text: str, max_tokens=750, overlap=120):
    # naive token-ish chunking by words
    words = text.split()
    step = max_tokens - overlap
    for i in range(0, len(words), step):
        yield " ".join(words[i:i+max_tokens])

def index_file(path: Path, meta: dict):
    raw = path.read_bytes()
    text = extract_text_any(path)  # implement via parsers; OCR fallback for
PDFs
    doc_id = str(uuid.uuid4())
    meta = {**meta, "doc_id": doc_id, "source_uri": str(path)}

    chunks, ids, metadatas = [], [], []
    for j, chunk in enumerate(chunk_text(text)):
        cid = str(uuid.uuid4())
        ids.append(cid)
        chunks.append(chunk)
        metadatas.append({**meta, "chunk_index": j})

    vectors = model.encode(chunks, normalize_embeddings=True).tolist()
    coll.add(documents=chunks, embeddings=vectors, metadatas=metadatas, ids=ids)

# Example usage
for f in Path("./docs").rglob("*"):
    if f.suffix.lower() in {".pdf", ".docx", ".pptx", ".xlsx", ".csv", ".txt"}:
        index_file(f, {"doc_type": "report", "acl": {"roles": ["planner"]}})
```

Query + Synthesis (pseudo)

```
def retrieve(query, k=8, filters=None):
    # apply ACL filters into vector store where possible
    res = coll.query(query_texts=[query], n_results=k, where=filters or {})
    return list(zip(res["ids"][0], res["documents"][0], res["metadatas"][0]))

SYSTEM = """
You are SupplyChainGPT. Answer with evidence from retrieved chunks. If
uncertain, say so. Cite each claim with [doc_title §section or page].
"""

def answer(query, user_context):
    filters = {"acl.roles": {"$in": user_context["roles"]}}
    hits = retrieve(query, k=8, filters=filters)
    context = "\n\n".join([f"[SOURCE {i+1}] {t}\nMETA: {m}" for i, (_, t, m) in
enumerate(hits)])
    prompt = f"{SYSTEM}\n\nQuestion: {query}\n\nContext:\n{context}\n\nAnswer:"
    return call_llm(prompt)
```

---

12) Access Control & Multitenancy

- **Row-level filtering** in the vector store via `where` metadata (tenant, role, user ID).
- Store ACL as allow lists. Example:

```
acl = { tenants: ["acme"], roles: ["planner","ops"], users: ["uid_123"] }
```

- UI: hide sources the user lacks permission to view; server still enforces filtering.
- For strict environments, store only **doc pointers** in the vector DB; fetch full text on demand from a secure content service.

---

13) Observability & Ops

- **Logs**: query text (hashed), retrieved IDs, latency, LLM token usage, user feedback.
- **Dashboards**: retrieval hit rate, citation click-through, doc coverage by type/time.
- **Data quality**: monitor OCR confidence, parser errors, empty pages.
- **CI tests**: eval suite runs on PR; blocks if Recall@10 drops > X%.

---

14) Roadmap (RAG Workstream)

Week 1–2: MVP ingestion (PDF/DOCX/XLSX), chunking, embeddings, Chroma index, simple UI.

Week 3: Hybrid search + re-ranker; add ACL; citations with page/section.

Week 4: PII redaction; evaluators; feedback loop in UI.

Week 5: Connect one enterprise source (SharePoint/Azure Blob). Scheduling via Airflow.

Week 6: Hardening (caching, rate limits), observability, README + demo video.

---

15) UX Notes for the Chat App

- Left: chat stream; Right: sources panel (expand to view chunk, open full doc, highlight matches).
- Quick filters above input: [Warehouse] [SKU] [Supplier] [Date range].
- Actions: “Export answer + sources to PDF/Email,” “Create task from recommendation.”
- Guardrail UI: badges like *Low evidence*, *Outdated doc*, *Policy conflict*.

---

16) Stretch Enhancements

- **Query-aware chunking** (ColBERT-style late interaction) for better token-level matching.
- **Multi-lingual** embeddings (English + Hindi) for India ops.
- **Toolformer** style routing: auto-invoke table reader for numeric queries.
- **Continual learning**: fine-tune embeddings model on click feedback.

Deliverable of this document: an implementation-ready blueprint for the RAG layer that you can hand to teammates or start coding against immediately.