Architectural Overview

Model

The Fintech RAG Agent uses **Groq's hosted Llama-3.3-70B-Versatile model** through the official Groq Python client.
This large-language model provides fast inference, strong factual grounding, and deterministic low-temperature completions suitable for enterprise retrieval-augmented tasks.
No other API keys or third-party services are required; the system runs entirely with a single GROQ_API_KEY.

Indexing and Retrieval

Instead of an external vector database, the agent employs a **BM25 lexical retriever** built with the rank-bm25 library.
During ingestion, the DOCX source file (Fintech_intake.docx) is parsed into semantically atomic text chunks, each annotated with its hierarchical section path.
All chunks are tokenized and indexed locally into a **BM25Okapi** object that is serialized to storage/bm25_index.pkl.
This design eliminates dependency on any external vector store while providing high precision for well-structured enterprise documents.

RAG Pipeline Steps

1. **Ingestion** – The .docx file is read and converted into normalized text chunks (UUID-labeled) saved as chunks.jsonl.

2. **Indexing** – BM25 builds a term-frequency inverse-document-frequency (TF-IDF) index for lexical retrieval.

3. **Retrieval** – For each user query, the top-K most relevant chunks are retrieved based on lexical similarity.

4. **LLM Re-Ranking** – The Groq Llama-3 model re-ranks these chunks by contextual relevance and selects the minimal subset needed to answer precisely.

5. **Answer Generation** – The model is instructed with strict guardrails to generate **plain-text only** answers supported exclusively by the selected context.

    - If information is missing, it responds: *"Not found in Fintech_intake.docx."*

    - The system omits all citations, IDs, or JSON wrappers to maintain a clean natural-language output.

6. **Conversation Handling** – The Flask API accepts an optional history array so that follow-up questions inherit context for conversational continuity.

Architecture Summary

Layer	Technology	Purpose
Frontend (UI)	HTML + Vanilla JS	User interface for asking questions and viewing plain-text answers
Backend API	Flask (Python 3)	Serves /chat endpoint and renders the UI
Retrieval	BM25Okapi index	Local, dependency-free document retrieval
Generation	Groq Llama-3.3-70B-Versatile	Factual, low-temperature text generation
Storage	JSONL + Pickle	Lightweight persistence for chunks and index
Output Format	Plain text only	Ensures compliance with precision and irrelevance-exclusion requirements

This architecture achieves a production-ready, Groq-only RAG pipeline that meets all precision, contextual relevance, and simplicity requirements while remaining fully auditable and reproducible.