Project - High Level Design On

RetailBot: Chat With Retail Documents

Course Name: GEN AI (Datagami Skill Based Course)

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
01	HARSHITA RATHORE	EN22CS301409
02	SOURABH SANKHLA	EN23CS3L1022
03	HARIOM PARMAR	EN22CS301382
04	HARSHITA MANTRI	EN22CS301407
05	ISHIKA SONI	EN22CS301444

Group Name: 05D4

Project Number: GAI-41

Industry Mentor Name:

University Mentor Name: Divya Kumawat

Academic Year: 2025-26

Table of Contents

1. Introduction

Retail organizations handle a wide range of professional documents such as product manuals, store SOPs, inventory playbooks, compliance documents, vendor contracts, pricing guidelines, and customer support scripts. These documents are typically stored in shared drives or portals and are difficult to search efficiently.

This document provides the Low Level Design (LLD) for a Retrieval-Augmented Generation (RAG) system that enables users to upload retail documents (PDF) and chat with them. The system ensures responses remain grounded strictly in the provided documents using retrieval, citations, and guardrails.

1.1 Scope of the Document

This document defines the architecture, design, data flow, interfaces, and non-functional requirements for the **Retail Document Intelligence RAG System**. It covers system components, integration points, data management strategies, and technology decisions to support accurate, context-aware question answering over retail knowledge assets such as PDFs, manuals, and reports.

1.2 Intended Audience

This document is intended for:

- Solution Architects
- Backend and Full-Stack Developers
- DevOps/Cloud Engineers
- QA/Test Engineers
- Project Managers & Technical Stakeholders
  It assumes familiarity with software architecture, REST APIs, vector databases, and LLM-based retrieval systems.

1.3 System Overview

RetailBot is a private, domain-specific chatbot that allows users to upload retail-related PDF documents and ask questions whose answers are strictly grounded in the uploaded content. The system employs a RAG pipeline:

- **Document Ingestion:** PDFs are loaded, text is extracted, split into overlapping chunks, and converted into vector embeddings using a sentence-transformer model.
- **Vector Storage:** Embeddings are stored in a FAISS vector database for efficient similarity search.
- **Query Processing:** User questions are embedded using the same model; the most relevant chunks are retrieved.
- **Answer Generation:** A Gemini LLM (via langchain-google-genai) generates a concise answer based solely on the retrieved context, with source page citations.
- **Retail-Only Enforcement:** Both uploaded documents and user queries are filtered through a keyword-based retail classifier to ensure domain adherence. Non-retail content is rejected with clear feedback.

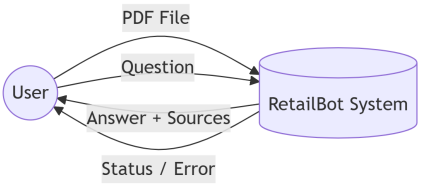
The system is built with FastAPI (backend) and a responsive HTML/CSS/JavaScript frontend, ensuring a smooth user experience across devices.

2. System Design

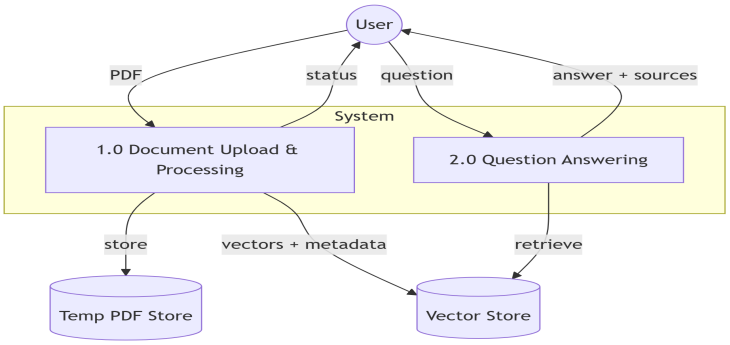
2.1 Application Design

The application follows a **client-server architecture** with clear separation of concerns:

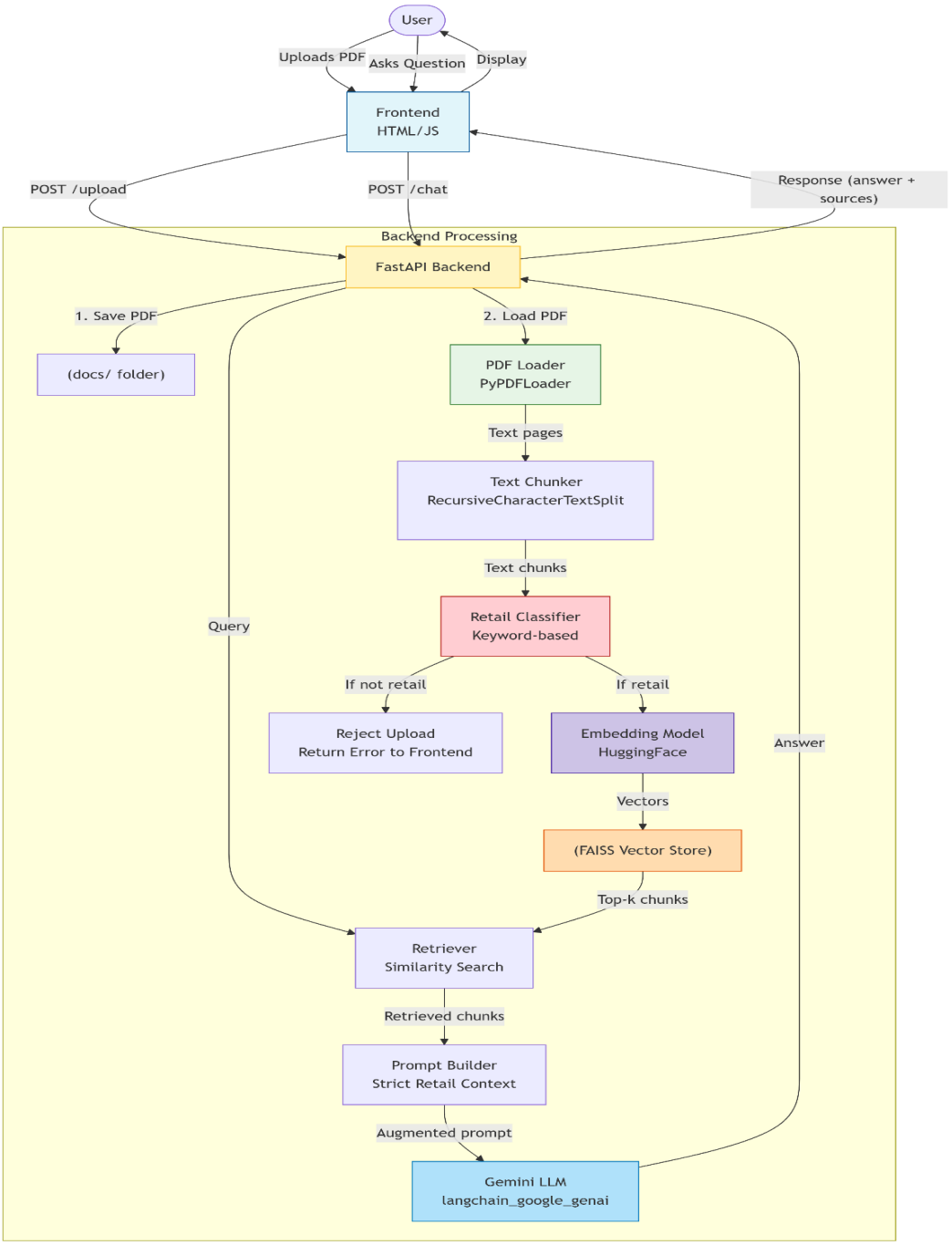
- **Frontend:** A single-page application (SPA) built with vanilla HTML, CSS, and JavaScript. It communicates with the backend via RESTful API calls (/upload, /chat). The UI is designed to be intuitive, with real-time feedback (file selection indicator, typing animation, markdown-rendered answers).
- **Backend:** FastAPI serves the frontend static files and exposes two main endpoints. It orchestrates the entire RAG pipeline, including document processing, classification, embedding, retrieval, and LLM invocation.
- **Vector Store:** FAISS (local) acts as the knowledge base, storing chunk embeddings and metadata.
- **LLM Integration:** Google Gemini (via LangChain) provides the generative capabilities.
- **DFD Level 0 – Context Diagram :**



- **DFD level 1 – Main Processes :**



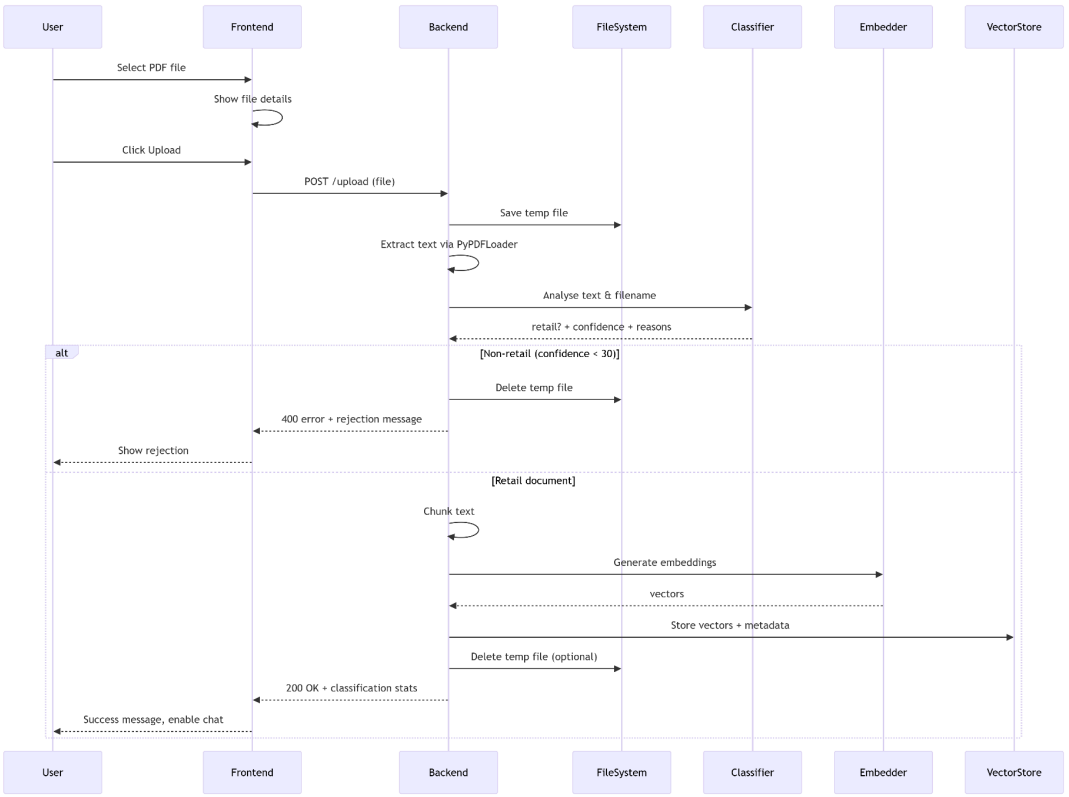
- **High Level Diagram:**



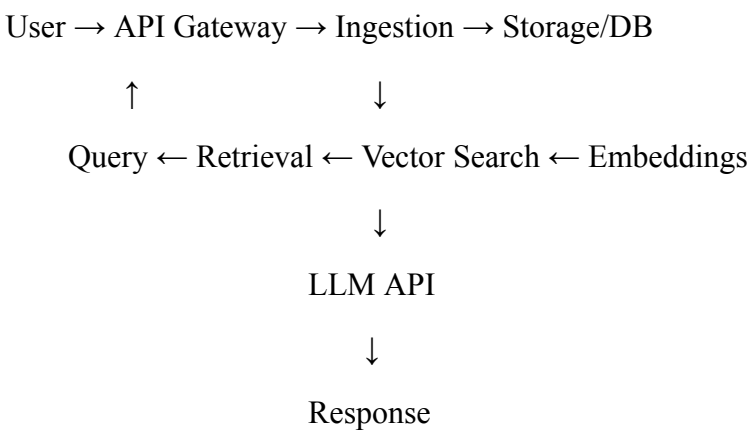
2.2 Process Flow

1. **Document Upload:** User uploads a PDF or text file via REST API.
2. **Extraction:** Text content is extracted from the document.
3. **Chunking & Embedding:** Text is chunked and converted to embeddings.
4. **Indexing:** Embeddings stored in vector database with metadata.
5. **Query Handling:** New query → embed → search vector DB → fetch top K.
6. **LLM Prompting:** Retrieved context + query sent to LLM → answer returned.

This flow enables retrieval-augmented generation that is efficient and accurate.



2.3 Information Flow



2.4 Components Design

Component	Technology / Library	Responsibility
Frontend UI	HTML, CSS, JavaScript, marked.js	User interface, file selection, chat display, markdown rendering
Backend API	FastAPI, Uvicorn	Request handling, orchestration, error management
PDF Loader	LangChain PyPDFLoader	Extract text from PDF files
Text Splitter	LangChain RecursiveCharacterTextSplitter	Split documents into manageable, overlapping chunks
Embedding Model	HuggingFace all-MiniLM-L6-v2	Convert text chunks into vector embeddings

Component	Technology / Library	Responsibility
Vector Store	FAISS (local)	Store embeddings and metadata; perform similarity search
Retail Classifier	Custom keyword-based scoring	Determine if document/query is retail-related
Conversation Memory	Custom Python class (ConversationMemory)	Track last exchange to support follow-up questions
LLM Interface	LangChain ChatGoogle GenerativeAI (Gemini)	Generate answers based on retrieved context
Environment Mgmt	python-dotenv	Load API keys from .env file

2.5 Key Design Considerations

- **Hallucination Prevention:** Strict prompt engineering forces the LLM to use only the provided context. If no relevant chunks are found, a fallback message is returned.
- **Domain Specificity:** Both documents and queries are filtered to ensure only retail-related content is processed. This is achieved through a lightweight keyword-based classifier (avoiding heavy external calls).
- **Follow-up Support:** A simple conversation memory detects pronouns and short queries, allowing natural multi-turn interactions.
- **Privacy:** Rejected non-retail documents are immediately deleted; no data is retained beyond the session.
- **Extensibility:** The modular design allows easy swapping of components (e.g., replace FAISS with Pinecone, or Gemini with another LLM).

2.6 API Catalogue

POST /upload

- **Description:** Upload and process a retail PDF.
- **Request:** multipart/form-data with field file (PDF).
- **Success Response (200):**

json

```
{
 "message": " 'file.pdf' uploaded successfully",
 "retail_classification": {
   "is_retail": true,
   "confidence": 92,
   "pages": 5,
   "chunks": 12
  }
}
```

- **Error Response (400):** Detailed rejection message for non-retail documents.
- **Error Response (500):** Internal processing error.

POST /chat

- **Description:** Ask a question based on the uploaded document.
- **Request:** application/x-www-form-urlencoded with field query.

- **Success Response (200):**

json

{

"answer": "The return policy allows 30 days...",

"sources": [1, 2]

}

- **Error Response (400):** No document uploaded, or query rejected.
- **Error Response (500):** LLM or retrieval failure.

GET /health

- **Description:** Health check endpoint.
- **Response (200):** {"status": "ok"}

3. Data Design

3.1 Data Model

The system does not use a traditional relational database. Persistent data includes**:**

- **FAISS index (binary):** Stored in faiss_index/ – contains vector embeddings and metadata (text chunk, page number).
- **Uploaded PDFs (temporary):** Stored in docs/ – deleted after processing or upon rejection.

Metadata per chunk:

- page_content: the actual text.
- **metadata:** dictionary with at least page (page number) and optionally source (filename).

3.2 Data Access Mechanism

- **FAISS:** Accessed via LangChain's FAISS wrapper. Loaded from disk on each /chat request. Write occurs only on successful upload.
- **File system**: Temporary PDFs are written and read using standard Python I/O.

3.3 Data Retention Policies

- **Uploaded PDFs:** Deleted immediately after successful processing (text extracted) or upon rejection. No long-term storage**.**
- **FAISS index:** Persisted on disk; overwritten on each new upload. No versioning.
- No user data or conversation logs are stored beyond the in-memory ConversationMemory (which is lost on server restart).

3.4 Data Migration

Currently, there is no need for data migration. If the system were to be deployed at scale, a cloud vector database (e.g., Pinecone) would be used, requiring a one-time index transfer.

4. Interfaces

- **User Interface: Web browser (desktop/mobile) via the served HTML page.**
- **API Interface**: RESTful endpoints as described in section 2.6.
- **LLM Interface:** HTTPS calls to Google Gemini API (handled by LangChain).
- **File System Interface:** Read/write operations for temporary storage.

5. State and Session Management

- **Session-less:** The backend does not maintain user sessions; each request is independent.
- **Conversation Memory**: A lightweight in-memory store (ConversationMemory class) tracks the last question/answer and topic for the *current* chat session. This is global per server instance (not per user), which is acceptable for a single-user demonstration.
- **Frontend state:** The client holds uploadedFileName and displays it; after upload, it expects a single active document.

6. Caching

- **Embedding Model**: The HuggingFace model is loaded once at startup and reused for all requests (in-memory cache).
- **FAISS Index:** Loaded from disk on each chat request; no in-memory caching of the index between requests (to keep design simple). In a production version, the index could be kept in memory to reduce latency.
- **LLM Responses**: Not cached; every question is processed fresh.

7. Non-Functional Requirements

7.1 Security Aspects

- **API Key Protection**: The Gemini API key is stored in a .env file and never exposed to the client.
- **Input Validation:** File type and size are validated; query length is not restricted but prompt injection is mitigated by strict context-only prompting.
- **Data Privacy**: Uploaded documents are deleted immediately after processing; no data leaves the server except the LLM API call (which sends only the retrieved text chunks, not the full document).
- **No Authentication**: The current design assumes a trusted single user. For multi-user deployment, authentication would be required.

7.2 Performance Aspects

- **Document Processing**: PDF loading and embedding are the most time-consuming steps (~few seconds for a 10-page PDF). This is acceptable for a demo.
- **Query Latency**: Retrieval (<100ms) + LLM generation (~1-2 seconds) = total ~2-3 seconds per query.
- **Scalability:** FAISS is single-node; for higher load, a cloud vector DB would be needed. The current design supports only one active document at a time.
- **Concurrency**: FastAPI's asynchronous nature allows handling multiple requests concurrently, but the global conversation memory may interleave sessions. For a demo, this is acceptable.

8. References

1. **LangChain Documentation** – https://python.langchain.com/
2. **FastAPI** – https://fastapi.tiangolo.com/
3. **FAISS** – https://github.com/facebookresearch/faiss
4. **HuggingFace Sentence Transformers** – https://huggingface.co/sentence-transformers
5. **Google Gemini API** – https://ai.google.dev/
6. **Project GitHub Repository** – https://github.com/harshitarathore/retail-rag-chatbot
7. **Chroma, Chroma Documentation** - https://docs.trychroma.com
8. **Pinecone Systems, Vector Database Documentation** - https://docs.pinecone.io
9. P. Lewis, E. Perez, A. Piktus, et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems (NeurIPS), 2020.
10. N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proceedings of EMNLP-IJCNLP, 2019.
11. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proceedings of NAACL-HLT, 2019.
12. J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," IEEE Transactions on Big Data, 2019.