# Contextual RAG with IR and Vector Embedding Retrieval

## High Level Components

1. **Frontend (Next.js Application)**

- **Role:** User Interface
- **Description:**
  The frontend is a web application built with Next.js. It allows users to upload documents, ask questions, view answers, and trigger evaluation. All user actions are sent as API requests to the backend. Though it was in the scope of document of I thought of developing to make the testing easy. There are lots of scope of improvements in terms of look and feel which I can do later if given additional time.

---

2. **Backend (FAST API Application)**

- **Role:** API Server and Core Logic
- **Description:**
  The backend is a FASTAPI-based API server responsible for handling all business logic, document processing, retrieval, LLM interaction, and evaluation.

2.1 **API Layer (app/api/)**

- **Files:** ingest.py, query.py, evaluate.py, ground_truth.py
- **Role:** Exposes RESTful endpoints to the frontend for:
  - Document ingestion for parsing and chunking
  - Querying the document with LLM
  - Running evaluation
  - Uploading ground truth data

2.2 **Core Logic (app/core/)**

- **Files:** chunker.py, contextaugmenter.py, evaluator.py, hybridretriever.py, llms.py, utils.py, vectordb.py
- **Role:** Implements the main functionalities:
  - **Chunker:** Splits documents into chunks for efficient retrieval (Two methods implemented)
  - **Context Augmenter:** Future Enhancement

- **Evaluator:** Calculates metrics like latency, recall, and similarity.
- **Hybrid Retriever:** Combines IR and vector search for better retrieval.
- **LLMs:** Interfaces with external LLM APIs for answer generation.
- **VectorDB:** Handles vector storage and similarity search (FAISS)
- **Utils:** Utility functions used across modules.

2.3 **Models (app/models/)**

- **Files:** schemas.py
- **Role:**
  Defines data models and schemas for API requests and responses, ensuring data consistency.

2.4 **Configuration**

- **File:** .env
- **Role:**
  Stores environment-specific settings (chunking strategy, model endpoints, API keys, etc.).

2.5 **Entry Point**

- **File:** main.py
- **Role:**
  Starts the Flask application and loads configuration.

---

3. **Data Storage (backend/app/data/)**

- **Subfolders:**
  - bm25_index_store: Stores BM25 index files for traditional IR.
  - faiss_vector_store: Stores vector embeddings for semantic search.
  - documents: Stores ingested documents.
  - ground_truth: Stores ground truth JSON files for evaluation.

---

4. **Test Data (testdata/)**

- **Role:**
  Contains sample documents (PDFs, JSON) for testing and evaluation.

---

5. **Supporting Files**

- **Dockerfile:**
  Defines the environment and dependencies for both backend and frontend, enabling containerized deployment.
- **README.md:**
  Project documentation.
- **LICENSE:**
  Licensing information.

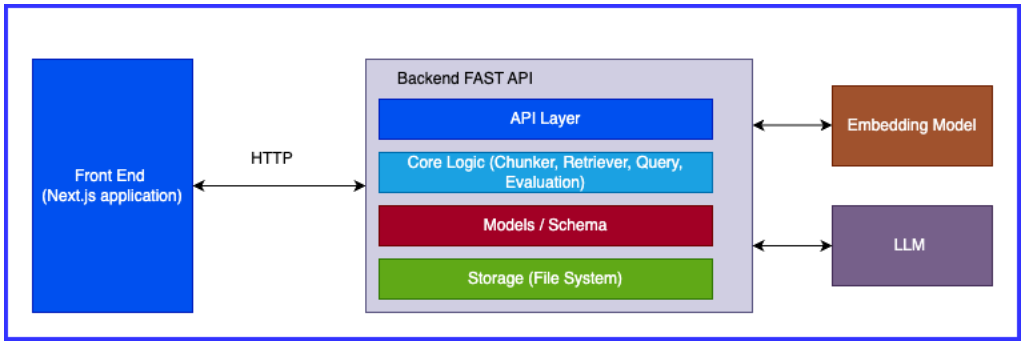
---

**Component Interaction Flow**

1. **User** interacts with the **Frontend** (uploads documents, asks questions, requests evaluation).
2. **Frontend** sends API requests to the **Backend** (FAST API).
3. **Backend API Layer** routes requests to the appropriate core logic module.
4. **Core Logic** handles document chunking, retrieval (BM25/Vector/Hybrid), LLM calls, and evaluation.
5. **Data** is read from and written to the appropriate storage directories.
6. **Responses** (answers, evaluation metrics) are returned to the **Frontend** for display to the user.

---

**System Design Overview**




**Important Design Decisions**

1. **Framework Choice: FastAPI**

- **Why FastAPI?**
  FastAPI is chosen for its speed, async support, automatic OpenAPI docs, and type safety. It enables rapid development. Swagger can be accessed by appending **/docs** to root URL

---

2. **Modular, Layered Backend Structure**

- **API Layer (app/api/)**
  Contains all route definitions (ingest.py, query.py, evaluate.py, ground_truth.py). Each file corresponds to a logical API area, following the **Single Responsibility Principle** (SRP) from SOLID.
- **Core Logic (app/core/)**
  Encapsulates business logic: chunking, context augmentation, retrieval, LLM and embedding handling, utilities, and vector DB operations. This separation ensures code is easy to test, maintain, and extend.
- **Models (app/core/models/)**
  Defines request/response schemas for strong typing and validation, leveraging Pydantic models.

---

3. **LLM and Embedding Model Strategies**

- **LLM: Claude Sonnet (claude-3-5-sonnet)**
  Used for answer generation, chosen for its strong language understanding and reasoning capabilities.
- **Embedding Model: BGE-M3 (text-embedding-bge-m3)**
  Used for semantic chunk embedding and vector search, offering high retrieval accuracy for multilingual and domain-specific queries.
- **Configurable via .env:**
  Model names, API keys, and endpoints are environment-configurable for flexibility and security.

---

4. **Chunking and Retrieval**

- **Chunking Strategies:**
  - Semantic Chunking: Chunks are determined using semantic boundaries (e.g., paragraphs, LLM-assisted segmentation).

    - Overlap Chunking: Fixed-size chunks with overlaps for context preservation.
- **Retrieval Strategies:**
    - **Hybrid Retrieval:** Combines BM25 (lexical) and vector (semantic) search for robust performance and combining based on score and rank.
    - **Pluggable Backends:** The design allows easy addition of new retrieval or chunking strategies.

---

5. **Data Organization**

- **Dedicated Data Directories:**
    - bm25_index_store/ for classic IR indexes.
    - faiss_vector_store/ for vector embeddings.
    - documents/ for raw and processed documents.
    - ground_truth/ for evaluation data.
- **Test Data:**
    - testdata/ for sample PDFs and ground truth JSONs.

---

6. **Incorporation of SOLID Principles**

- **SOLID:**
    - **Single Responsibility:** Each module/class does one thing (e.g., chunking, retrieval).
    - **Open/Closed:** New retrieval or chunking methods can be added without modifying existing code.
    - **Liskov Substitution:** Retrieval and chunking interfaces allow easy swapping of implementations.
    - **Interface Segregation:** Only relevant methods are exposed in each module.
    - **Dependency Inversion:** Core logic depends on abstractions, not concrete implementations.

---

7. **Scalability**

- **Stateless API:**
    - FastAPI backend is stateless and horizontally scalable; can be containerized and deployed across multiple nodes.
- **Separation of Concerns:**
    - Frontend and backend can scale independently.
- **Pluggable Storage:**
    - Current design supports local storage, but can be extended to distributed vector DBs (Milvus, PGVector etc).
    - Documents can be stored on S3 or any external storage as future enhancements
- **Async & Batch Processing:**
    - FastAPI supports async endpoints for high concurrency.

---

8. **Future Improvements**

- **Vector Database Integration:**
    - Support for scalable vector stores like **Milvus** or **pgvector** for billions of embeddings and distributed search.
- **Enhanced Chunking:**
    - Use LLMs to add richer context to each chunk, improving answer relevance as explained in Anthropic Paper
- **Multiple Document Support:**
    - Ingest and query across large collections, with metadata filtering.
- **Advanced Parsing:**
    - Implement **RAPTOR** or **Hierarchical Node Parsing** from llama_index for better context understanding and answer granularity.
- **Document Versioning & Metadata:**
    - Track versions and sources for traceability and auditability.
- **Streaming & Real-Time Updates:**
    - Enable real-time document ingestion and retrieval.

- **User Authentication & Access Control:**
    - Add user management and permissions for enterprise use cases.