□ AI Document Assistant – RAG-Based Architecture

□ Overview

This assistant enables:

- Ingesting and processing documents from a local `data/` directory

- Chunking and embedding content using HuggingFace Transformers

- Storing and retrieving document embeddings via Supabase Vector Store

- Contextual Q&A with multi-turn conversations using MongoDB-based memory

- Supporting multiple LLMs (both online and offline) via a pluggable architecture

□ Assumptions & Prerequisites

1. Documents (PDF, DOCX, TXT) reside in a local `data/` folder.

2. Embeddings are generated once and stored in Supabase Vector DB.

3. Session-based memory is maintained in MongoDB Atlas.

4. Application supports both online (e.g., Cohere, Mistral via APIs) and offline (e.g., LLaMA3, DeepSeek via Ollama) LLMs.

5. Codebase is modular and supports:

   - Pluggable file parsers

   - Swappable LLM providers

- Extensible storage (vector/memory)

□ Document Processing Pipeline

Run as a preprocessing script.

Pipeline Steps:

1. Read files in `data/` folder

2. Parse using format-specific parser (PDF, DOCX, TXT, etc.)

3. Chunk text into semantically meaningful blocks

4. Embed chunks using HuggingFace model: sentence-transformers/all-MiniLM-L6-v2

5. Store embeddings with metadata into Supabase Vector Store

□ Built using the Factory Pattern, allowing easy extension for new file types.

□ Question Answering Flow

Endpoint: POST /ask

Example Request:

```
{
"isOnline": true,
"llm": "mistral",
"question": "What is our leave policy?",
"sessionId": "abc123"
}
```

Internal Workflow:

1. Retrieve relevant documents from Supabase using similarity search.

2. Fetch memory from MongoDB for the given sessionId.

3. Assemble prompt:

   - Context from retrieved docs

   - Chat history

   - Current question

4. LLM Strategy Selection via Factory:

  - Choose between online (e.g., OpenAI, Cohere) and offline (e.g., Ollama) models

5. Generate answer and update memory store with user & assistant messages.

□ Memory Management

LangChain’s BufferMemory used for chat memory.

Backed by MongoDB Atlas.

Each session identified via sessionId.

Memory APIs:

- GET /history/:sessionId – Fetch chat history

- GET /sessions – List all sessions and their titles (first user message)

□ Design Patterns Used

⍰ Factory Pattern

Used for dynamic selection of:

- LLMs: LLMFactory.create(isOnline, llmName)

- File parsers: ParserFactory.getParser(fileType)

⍰ Strategy Pattern

Each LLM implements a common interface:

```
interface LLMStrategy { generate(prompt: string): Promise<string>; }
```

Examples: OpenAIModel, OllamaModel, CohereModel, DeepSeekModel

□ Supported LLMs

✓Online (via APIs):

- mistral, cohere.

✓Offline (via Ollama @ http://localhost:11434):

- llama3, deepseek.

Example for Offline Usage:

```
{
"isOnline": false,
"llm": "llama3"
}
```

Adding a new model:

- Implement the LLMStrategy interface

- Register it in LLMFactory

□ Extensibility

Feature	How to Extend
New LLMs	Add a new Strategy class and register it
New file formats	Add a new parser and register in ParserFactory
Vector DB	Replace SupabaseVectorStore with another provider
Prompt Logic	Customize LangChain prompt template configuration

□ Scalability Considerations

- Modular code structure

- Easy provider swap (LLM, parser, memory, vector store)

- Supports async ingestion and batch processing

- Embedding metadata (e.g., timestamps, file sources) for audit or analytics

- Ready for deployment in cloud-native environments