Technical Assessment: Smart Document Q&A Assistant

Overview

Build a FastAPI application with a Next.js frontend that allows users to upload documents (PDF/TXT) and ask questions about them using RAG (Retrieval-Augmented Generation) powered by LangChain and vector embeddings.

Requirements

API Endpoints

Your FastAPI application should implement the following endpoints:

1. **POST /api/documents/upload** - Accept document upload (PDF or TXT format)
2. **GET /api/documents** - Retrieve list of all uploaded documents
3. **POST /api/documents/query** - Ask questions about a specific document
4. **DELETE /api/documents/{document_id}** - Delete a document (Optional)

RAG Pipeline Implementation

Implement a complete RAG (Retrieval-Augmented Generation) system:

1. **Document Processing**
   - Extract text from uploaded PDF/TXT files
   - Split text into chunks (recommended: 500-1000 characters with 50-100 character overlap)
   - Generate embeddings for each chunk using OpenAI or HuggingFace embeddings
2. **Vector Storage**
   - Store embeddings in FAISS vector database

- Persist FAISS index to disk for retrieval
- Store document metadata in PostgreSQL

3. **Question Answering**
   - Perform similarity search to retrieve relevant chunks
   - Use LangChain to construct prompts with retrieved context
   - Generate answers using LLM (OpenAI GPT-3.5/4, Groq, or similar)
   - Return answer with source references

Technical Requirements

- Use **FastAPI** framework for backend
- Use **Next.js** (with App Router or Pages Router) for frontend
- Implement **LangChain** for RAG pipeline orchestration
- Use **FAISS** for vector storage and similarity search
- Use **PostgreSQL** for storing document metadata
- Use any LLM provider: OpenAI, Groq (free tier), or HuggingFace
- Handle file uploads with proper validation (file size, type)
- Implement proper error handling for document processing and LLM failures
- All API responses must be in **JSON format**
- Frontend should be **responsive** (mobile and desktop)

Expected API Response Formats

Upload Document Response:

```
{ "document_id": "abc123", "filename": "research_paper.pdf", "status":
"processed", "chunks_created": 45, "uploaded_at": "2024-10-24T10:30:00Z" }
```

Query Document Response:

```
{ "document_id": "abc123", "question": "What are the main findings?",
"answer": "The main findings indicate...", "sources": [ { "chunk_text":
"...relevant excerpt...", "relevance_score": 0.92 } ],
"processing_time_seconds": 2.3 }
```

List Documents Response:

```
{ "documents": [ { "document_id": "abc123", "filename": "research_paper.pdf", "uploaded_at": "2024-10-24T10:30:00Z", "chunk_count": 45 } ] }
```

Frontend Requirements

Pages/Components to Build

1. **Home/Upload Page**
   - File upload form (drag-and-drop optional)
   - Display list of uploaded documents
   - Show upload progress/status
   - Delete button for each document (optional)
2. **Q&A Interface**
   - Document selector (dropdown or list)
   - Question input field
   - Submit button
   - Display area for:
     - Question asked
     - AI-generated answer
     - Source references/chunks
   - Loading state while processing query

UI/UX Requirements

- Clean, modern design using Tailwind CSS or shadcn/ui
- Responsive layout (mobile-first approach)
- Loading spinners/skeletons during async operations
- Error messages displayed to user in a friendly manner
- Toast notifications for successful uploads/actions

Deliverables

1. GitHub Repository containing:

- Complete FastAPI backend code
- Complete Next.js frontend code
- README.md with:
  - Project description
  - Setup instructions (step-by-step)
  - API documentation with example requests/responses
  - Design decisions explanation (max 500 words)
  - Screenshots of working application
- requirements.txt (Python dependencies)
- package.json (Node dependencies)
- .env.example file showing required environment variables

2. Demo Video (2-3 minutes):

- Upload a sample document
- Ask 2-3 questions and demonstrate answers
- Brief walkthrough of RAG implementation in code

3. Sample Test Documents

- Include at least 2 sample documents (PDF/TXT) for testing
- Examples: research paper, article, documentation

Bonus Features (Optional)

Implement any of these for extra credit:

- 🌟 Support for multiple file formats (DOCX, Markdown)
- 🌟 Conversation history (show previous Q&A pairs for a document)
- 🌟 Answer streaming (display answer as it generates in real-time)
- 🌟 Citation highlighting (show which parts of the answer came from which chunks)
- 🌟 Document preview/viewer in the UI
- 🌟 Deployed application (Vercel for Next.js, Railway/Render for FastAPI)
- 🌟 Simple authentication (username/password)

- 🌟 Cost optimization (caching frequent queries, batch processing)

Helpful Resources & Tips

LLM Provider Options (Choose One)

- **Groq** - Very fast, generous free tier (Recommended for this project)
- **OpenAI** - New accounts get $5 free credit
- **HuggingFace** - Completely free but slower

Key Libraries to Use

- **Backend:** fastapi, langchain, langchain-community, faiss-cpu, pypdf2 (or pdfplumber), sqlalchemy, psycopg2-binary
- **Frontend:** next, react, axios, tailwindcss, lucide-react (for icons)

Testing Your Application

Use documents like:

- Wikipedia article saved as PDF
- Short story or essay (TXT)
- Technical documentation or research paper

Questions?

If you have any clarifications needed regarding:

- LLM provider selection or API key setup
- Expected behavior for specific edge cases
- Technical constraints or library versions

Please reach out before starting the implementation.

Good luck!