Frontend/Backend Assignment - Document Intelligence Platform

To grab the internship, you are required to complete a small assignment. Early submissions are given preference.

In this task, you are required to create a full-stack web application with AI/RAG integration.

Objective:

The goal of this assignment is to build a Document Intelligence Platform where users can upload documents (PDFs, Word docs, text files) and ask natural language questions about the content. The system should use RAG (Retrieval Augmented Generation) to provide accurate, contextual answer. Only the tech-stack mentioned in the assignment should be used.

Backend (Django REST Framework)

GET APIs:

Create API endpoints that:

- Retrieve all uploaded documents from the database

- Get document metadata and processing status

POST APIs:

Create API endpoints for:

- Uploading and processing documents (PDF, DOCX, TXT)

- Asking questions about documents (RAG query endpoint)

DELETE API:

Create an endpoint to delete documents and associated data.

You may add any other APIs if needed.

Document Processing Engine

Create a Python module that handles document processing and RAG implementation:

Extra marks for handling multiple document formats, smart chunking strategies, and optimized embedding generation.

The system should:

- **Document Parsing**: Extract text from PDFs, Word documents, and text files
- **Text Chunking**: Split documents into meaningful chunks (paragraphs, sections)
- **Embedding Generation**: Create vector embeddings for document chunks
- **Vector Storage**: Store embeddings in a vector database (ChromaDB or FAISS)
- **Similarity Search**: Find relevant chunks based on user queries
- **Answer Generation**: Use OpenAI/Anthropic API/LM Studio to generate contextual answers

Core Features:

- Document text extraction and preprocessing
- Intelligent text chunking with overlap
- Vector embedding generation using sentence transformers
- Semantic search across document chunks
- Context-aware answer generation

Input Parameters for Questions:

- Document ID
- User question
- Chat session ID (optional)
- Number of relevant chunks to retrieve (default: 3-5)

RAG Pipeline Implementation:

The system should implement a complete RAG pipeline that:

- Generates embeddings for user questions

- Performs similarity search across document chunks

- Constructs relevant context from retrieved chunks

- Generates contextual answers using LLM with source citations

Frontend (ReactJS / NextJS with Tailwind CSS)

User Interface:

The frontend should be developed using ReactJS or NextJS, styled with Tailwind CSS.

The interface should provide:

- **Document Upload Page**: Drag-and-drop interface for uploading documents
- **Document Library**: Display all uploaded documents with metadata
- **Chat Interface**: Q&A interface for each document

Required Pages:

1. **Dashboard/Library Page**:
   - List all uploaded documents
   - Basic document metadata (title, pages, upload date)
2. **Document Chat Page**:
   - Chat interface (right panel)
   - Question input
   - Answer display
3. **Upload Page**:
   - File upload interface
   - Document preview after upload (optional)

Technical Requirements:

- Use React hooks for state management
- Implement file upload
- Add loading states and error handling
- Implement proper error boundaries

Database Schema Hints:

Design your database with the following tables structure:

Documents table: Store document metadata including title, file path, type, size, pages, processing status, and timestamps.

Document chunks table: Store processed text chunks with references to parent document, chunk index, page numbers, and embedding identifiers for vector database integration.

Chat sessions table: Manage conversation sessions linked to specific documents with creation timestamps.

Chat messages table: Store question-answer pairs with source references and metadata for each chat interaction.

Tech Stack Requirements:

- **Backend**: Django REST Framework, Python
- **Database**: MySQL for metadata, ChromaDB/FAISS for vectors
- **Document Processing**: PyPDF2/pdfplumber, python-docx, sentence-transformers
- **AI Integration**: OpenAI API, Anthropic Claude API, or **LM Studio** (recommended if external APIs are not available)

- **Frontend**: ReactJS/NextJS with Tailwind CSS
- **File Storage**: Local storage

AI Integration Options:

Option 1: Use external APIs (OpenAI, Anthropic Claude)

Option 2: Use LM Studio for local LLM hosting (recommended alternative)

LM Studio allows you to run language models locally without requiring external API keys. Download and set up LM Studio with models like Llama, Mistral, or Code Llama for document question-answering tasks. This provides a cost-effective solution and ensures data privacy.

Deadline

31 May, 2025, Friday, 11:55 PM.

Submission Format

- Create a GitHub repository and add your code to it.
- In the README, add:
    - Screenshots of the UI you have created
    - Setup instructions for running the application
    - API documentation
    - Sample questions and answers from your system
- Include a requirements.txt file with all dependencies
- Add sample documents for testing (PDFs, Word docs)
- Fill your repo link in this form: https://forms.gle/ZT6VZ1iW3ah91Gxu5.
- Make sure the code is neat and readable with proper comments.

Evaluation Criteria:

- **Functionality** (40%): Working RAG pipeline, accurate answers, proper citations
- **Code Quality** (25%): Clean, readable, well-structured code
- **UI/UX** (20%): User-friendly interface, responsive design
- **Innovation** (15%): Creative features, optimization techniques, error handling

Bonus Points:

- Support for multiple document formats (PDF, DOCX, TXT, MD)
- Advanced chunking strategies (semantic chunking, overlapping windows)
- Real-time processing status updates
- Document highlighting showing relevant sections
- Export functionality for chat history
- Advanced search and filtering capabilities

- Performance optimizations for large documents

For any doubts, feel free to drop a mail at devgods99@gmail.com We'll try to respond as soon as possible!

In case you're not able to complete it within the deadline, do submit the code even if a part of it doesn't work. The effort and skills you demonstrate through your code matter.

Note: An early submission will give you an advantage over others.

Helpful Resources:

- OpenAI API Documentation: https://platform.openai.com/docs
- LM Studio: https://lmstudio.ai/ (for local LLM hosting)
- ChromaDB Documentation: https://docs.trychroma.com/
- Sentence Transformers: https://www.sbert.net/
- Django REST Framework: https://www.django-rest-framework.org/
- React File Upload: https://react-dropzone.js.org/