FULL STACK GEN AI ENGINEER ASSIGNMENT

Technical Assignment: Insurance Policy Q&A with RAG

Objective: Build a monolithic web application using React (Frontend) and FastAPI (Backend) that allows users to upload insurance policy documents (PDF/Text) and query them via a Retrieval-Augmented Generation (RAG) system.

Core Requirements

1. Frontend (React)

- **Document Upload**:
    - File picker supporting PDF/TXT formats.
    - Display upload status (progress/error/success).
- **Chat Interface**:
    - Input field for user questions (e.g., *"What is the deductible for flood damage?"*).
    - Display AI responses in a conversational format.
- **Sources Tab**:
    - Show extracted document snippets used to generate the answer (highlight relevant text).

2. Backend (FastAPI)

- **Document Processing**:
    - Extract text from uploaded documents.
    - Chunk text logically with overlap handling.
- **RAG Pipeline**:
    - Generate embeddings for chunks using any model.
    - Store embeddings + chunks in a local vector database.
    - Retrieve relevant chunks for user queries using similarity search.
    - Generate answers using any LLM of your choice (Open source local or cloud commercial).
- **API Endpoints**:
    - POST /upload: Accept documents → return processing status.
    - POST /chat: Accept user query → return answer + sources.

3. Additional Requirements

- **Error Handling**:
    - Handle unsupported file types, LLM rate limits, and empty queries gracefully.
- **Basic UI/UX**:
    - Responsive design (mobile/desktop).
    - Loading states for AI responses.
- **Documentation**:
    - A README.md with setup instructions, design choices, and tradeoffs.

Evaluation Criteria

Read the below criteria carefully

Category	Expectations
Functionality	All core features work (upload, chat, sources). No critical bugs.
RAG Implementation	Logical chunking, accurate similarity search, and coherent answers.
Code Quality	Modular, readable, and well-commented. Proper error handling.
UI/UX	Intuitive interface with clear feedback (e.g., loading states).
Documentation	Clear setup steps and rationale for tech choices (e.g., "Chose FAISS for speed").

Submission Guidelines

1. **Code**: Push to a GitHub repository with a clean commit history.

2. **Deployment**: e.g., Vercel for frontend, Render for backend.
3. **Demo**: A 5-minute Loom video walking through:
    - Setup process.
    - Demo of all features.
    - Explanation of key technical decisions.

Timeline

- **Duration**: 1 week.
- **Late Submissions**: Penalized unless communicated in advance.

Tech Stack Recommendations

- **Frontend**: React + TypeScript, TailwindCSS/MUI.
- **Backend**: FastAPI, Python 3.10+.
- **Embeddings**: Of your choice.
- **Vector DB**: Of your choice.
- **LLM**: Local (preferred) or cloud (OpenAI GPT-4o).

FAQ

Q: Can I use Next.js instead of plain React?
A: Yes, but justify why.

Q: Is Docker/CI-CD required?
A: Yes. Bonus points for that.

Q: How large should the policy documents be?
A: Test with 5–50 page PDFs

We are looking for candidates to show off their

- Full-stack development and integration skills.
- Pragmatic AI/ML implementation skills.
- Attention to user-facing details.
- CI-CD skills