**Internal Test Project: Context-Aware Chat Application**

**Project Overview**

Build a simple chat application that can answer questions based on a provided PDF document. This is a 3-day technical assessment designed to evaluate your approach to API design, frontend development, and context handling.

**Core Requirements**

**1. Frontend (React)**

* **Single page chat interface** with:
  + Message input field
  + Send button
  + Chat history display (user and AI messages)
  + Clear/reset conversation button
* **Clean, responsive design** (can use any CSS framework or plain CSS)
* **Real-time streaming** of AI responses (bonus: show typing indicator)
* Messages should be clearly distinguished (user vs AI)

**2. Backend (FastAPI)**

* **Single endpoint** for chat messages: POST /chat
* **PDF context loading** on startup sera entregado por separado
* **Integration with OpenAI or Anthropic API** (API keys will be provided)
* **Streaming response support** using Server-Sent Events (SSE)
* **In-memory conversation history** (no database required)
* **Health check endpoint**: GET /health

**3. Context Handling**

* Load and parse the provided PDF on server startup
* Include the PDF content as context in all AI API calls
* Ensure the AI can accurately answer questions about the PDF content

**Technical Specifications**

**API Structure**

# Request

POST /chat

{

    "message": "What is the main topic of the document?",

    "conversation\_id": "optional-session-id"  # for maintaining context

}

# Response (SSE stream)

data: {"type": "content", "content": "The document discusses..."}

data: {"type": "done"}

**Required Dependencies**

* **Frontend**: React, axios/fetch for API calls
* **Backend**: FastAPI, python-multipart, PyPDF2 or similar for PDF parsing
* **AI Integration**: anthropic or openai Python SDK
  + OpenAI:
  + Anthropic:

**Evaluation Criteria**

**Code Quality (40%)**

* Clean, readable code with proper naming conventions
* Appropriate error handling
* Modular structure and separation of concerns
* Comments where necessary

**Functionality (30%)**

* All core features working as specified
* Smooth user experience
* Proper streaming implementation
* Accurate responses based on PDF context

**API Design (20%)**

* RESTful principles
* Proper request/response structure
* Error handling and appropriate status codes
* Clean endpoint design

**Frontend Design (10%)**

* Intuitive user interface
* Responsive design
* Proper state management
* Good UX practices

**Bonus Points (Optional Enhancements)**

If you complete the core requirements early, consider adding:

1. **Multiple conversation support** (switch between different chat sessions)
2. **Token/cost tracking** display for API usage
3. **Export conversation** to text/markdown
4. **Loading states** and better error handling UI
5. **Simple rate limiting** on the backend
6. **Markdown rendering** for AI responses

**Deliverables**

1. **GitHub repository** with:  
     
   * Clear README with setup instructions
   * Requirements files (package.json, requirements.txt)
   * .env.example file showing required environment variables
2. **Working application** that can be run locally with simple commands:

# Backend

cd backend

pip install -r requirements.txt

python main.py

# Frontend

cd frontend

npm install

npm start

1. **Brief documentation** explaining:  
     
   * Your design decisions
   * Any challenges faced and solutions
   * What you would improve given more time

**Timeline**

* **Day 1**: Backend implementation and AI integration
* **Day 2**: Frontend implementation and integration
* **Day 3**: Polish, testing, documentation, and optional features

**Getting Started**

You will be provided with:

1. A PDF file to use as context
2. API keys for OpenAI/Anthropic (choose one)
3. This specification document

**Tips for Success**

* Start simple and iterate
* Focus on core functionality first
* Test the context handling thoroughly
* Keep the code clean and well-organized
* Don't over-engineer - this is a simple chat app
* Ask questions if requirements are unclear

**What We're Looking For**

* How you structure your code
* Your approach to problem-solving
* Attention to user experience
* Understanding of API design principles
* Ability to integrate third-party services
* Code quality and maintainability

**Note**: This is not about building the most feature-rich application. We want to see clean, working code that demonstrates your understanding of full-stack development principles.