Full Stack AI Developer Take-Home Assignment

Time Estimate: 2-3 days
Stack: Next.js 14+ (App Router) + Python (FastAPI)

Project: AI Search Chat with PDF Citation Viewer & Generative UI

Build a Perplexity-style chat interface where AI responses are streamed with generative UI components, citations, and tool calls. Citations can be clicked to elegantly transition to a PDF viewer showing the highlighted source section.

Requirements

Core Features

1. **Chat Interface (Perplexity-style)**

- Clean, centered chat layout
- Each message shows:
  - User query
  - AI response with real-time streaming effect
  - Generative UI components (charts, cards, tables) streamed alongside text
  - Numbered citations `[1]`, `[2]`, etc. inline within the response
  - Source cards below each response showing cited documents
  - Tool call indicators showing reasoning steps (e.g., "Searching documents...", "Analyzing results...")

2. **Streaming Response with Generative UI**

- Implement Server-Sent Events (SSE) for real-time streaming
- Stream multiple types of content:
  - **Text chunks**: Incremental AI response text
  - **Tool calls**: Show reasoning steps (e.g., `thinking`, `searching_documents`, `retrieving_pdf`, `analyzing_content`)
  - **UI components**: Stream React components (charts, data tables, info cards) that render progressively [Optional]
  - **Citations**: Inline citation references that appear as the response generates
- Show typing indicator before streaming starts
- Display step-by-step reasoning as tool calls execute

3. **PDF Viewer with Entrance Transition**

- When user clicks a citation [1], [2], etc.:
  - Smooth, animated transition from centralized chat interface to split-view layout (chat + PDF viewer)
  - PDF Viewer features:
    - Display the source PDF document
    - Automatically highlight and scroll to the relevant section
    - Navigation controls (page up/down, zoom in/out, search)
    - Close button to return to chat-only view with reverse animation
  - Maintain chat scroll position during transition

4. **UI/UX Requirements**

Chat Interface:

- Perplexity-inspired design (clean, minimal, focused)
- Responsive layout (mobile-first)
- Citations styled as clickable, numbered badges [1]
- Source cards at bottom of each response with document metadata
- Loading states:
  - Typing indicator with animated dots
  - Streaming cursor/pulse effect
  - Tool call progress indicators (e.g., "🔍 Searching...", "📄 Reading PDF...")
- Generative UI components with smooth fade-in animations

PDF Viewer Transition:

- Entry animation: Smooth slide-in from right with scale/fade effect (300-400ms)
- Exit animation: Reverse of entry animation
- Responsive: On mobile, PDF viewer takes full screen; on desktop, split view (60/40 chat/PDF)

Technical Requirements

Frontend (Next.js 14+)

- **Next.js 14+** with App Router
- **TypeScript** with strict typing
- **Framer Motion** for animations (transitions, generative UI component reveals) [Optional]
- **react-pdf** or **@react-pdf-viewer/core** for PDF rendering [Optional]
- **Tailwind CSS** for styling

- **Zustand** for global state management (chat history, PDF viewer state) [Optional, Preferred]
- **TanStack Query (React Query)** for API requests and SSE handling [Optional, Preferred]

Backend (Python + FastAPI)

- **FastAPI** for REST API and SSE endpoints
- **Python 3.11+**
- **Pydantic** for request/response validation
- **Queue System** for request management [Optional, Preferred]
    - Use **Redis Queue (RQ)** / **Celery** / Other for asynchronous job processing
    - OR implement in-memory queue with **asyncio.Queue**
    - Enqueue generation requests to handle concurrency and rate limiting
    - Return job ID immediately, stream results via SSE as they're processed
- **PDF Processing**:
    - Use **PyPDF2** or **pdfplumber** for text extraction
    - Store PDF metadata and page-to-text mappings

Deliverables

1. **GitHub Repository** with:

- `/frontend` - Next.js application
- `/backend` - FastAPI application
- `docker-compose.yml` - For running the full stack (optional but recommended)

2. **README.md** including:

- **Setup Instructions**:
    - Backend setup (Python environment, dependencies, queue system)
    - Frontend setup (Node.js, dependencies)
    - Environment variables required
    - How to run locally (dev mode)
- **Architecture Overview**:
    - Diagram showing frontend ↔ backend ↔ queue flow
    - Explanation of streaming protocol
- **Screenshots/GIF**:
    - Tool call streaming in action
    - Generative UI components rendering
    - Citation → PDF viewer transition
- **Libraries Used**:
    - Complete list with versions and justification
- **Design Decisions**:

- Why you chose your queue system
- How you implemented generative UI
- Trade-offs made due to time constraints

3. **Code Quality Requirements**

- **TypeScript**: Strict typing, no `any` types
- **Python**: Type hints, Pydantic models
- **Error Handling**: Graceful failures, user-friendly error messages
- **Comments**: Complex logic should be documented
- **File Structure**: Organized, logical separation of concerns

---

Evaluation Criteria

1. **Streaming Implementation** (25%)

    - Smooth, real-time text streaming
    - Tool calls display correctly [Optional]
    - Generative UI components render progressively [Optional]

2. **PDF Viewer & Transitions** (20%)

    - Smooth animations (Framer Motion) [Optional]
    - Accurate citation highlighting

3. **Backend Architecture** (20%)

    - Queue system properly implemented [Optional]
    - SSE streaming works reliably
    - Clean API design

4. **Code Quality** (20%)

    - TypeScript/Python best practices
    - Proper error handling
    - State management (Zustand + React Query) [Optional]

5. **UI/UX Polish** (15%)

    - Perplexity-style aesthetic
    - Loading states and feedback
    - Responsive design [Optional]

Bonus Points (Optional)

- **Docker Setup**: Containerized app with docker-compose
- **Dark Mode**: Theme switcher
- **PDF Text Search**: Highlight search terms in PDF viewer

Good luck! Focus on clean implementation over feature completeness. A well-executed subset of features is better than a buggy full implementation.

References:

https://research.google/blog/generative-ui-a-rich-custom-visual-interactive-user-experience-for-any-prompt/
https://docs.langchain.com/langsmith/generative-ui-react
https://ai-sdk.dev/docs/ai-sdk-ui/generative-user-interfaces
https://docs.copilotkit.ai/
https://docs.langchain.com/oss/python/langgraph/streaming