# PlanSync AI

## Technical Coding Assessment

Software Engineering Intern Position

---

Time Limit: **3 Hours** — Submission: GitHub Repository

## 1 Overview

**Context**

PlanSync AI is a platform that helps retirement planning advisors manage 401(k) plan data. A core feature is our **AI-powered document extraction system** that processes plan documents (PDFs with 50-200 pages containing tables, forms, and legal text) and extracts structured data like plan names, contribution limits, and vesting schedules.
Your task is to build a **simplified version** of this extraction pipeline with a working backend and frontend.

## 2 The Challenge

Build a full-stack application that:

1. Accepts a PDF file upload
2. Extracts specific data fields (Hint: You can use some of AI-based tech stacks out there like LLMs, OCR, etc.)
3. Displays the extracted data in an editable UI, if there are no data notify about no data being found for a certain field and saves them to a local sql database

# 3 Technical Requirements

**Requirements**

## Backend (Python)

- Use **FastAPI** as the web framework
- Implement a `POST /extract` endpoint that:
  - Accepts one or multiple PDF file upload
  - Extracts text from the PDF (use `PyPDF2`, `pdfplumber`, or similar)
  - Sends the text to an LLM to extract the following fields:
    - **Plan Name**
      The official name of the retirement plan (e.g., "Acme Corp 401(k) Plan"). Usually found near the beginning of the document or in the General Information section.
    - **Employer Name**
      The legal name of the company sponsoring the plan. Often listed under "Employer Information" and may differ slightly from the plan name.
    - **Plan Effective Date**
      The date the plan originally became effective. Do not confuse this with amendment or restatement dates.
    - **Eligibility Requirements**
      A short text description of who can participate in the plan, typically including minimum age and service requirements.
    - **Employee Contribution Limit (%)**
      The maximum percentage of compensation an employee is allowed to defer into the plan, as stated in the document.
    - **Employer Match Formula**
      A text description of how employer matching contributions are calculated (e.g., "100% of the first 3% of compensation plus 50% of the next 2%").
    - **Safe Harbor Status**
      Indicates whether the plan includes Safe Harbor contributions. Return `Yes` if Safe Harbor contributions are explicitly permitted; otherwise return `No`.
    - **Vesting Schedule**
      A text description of how employer contributions vest over time (e.g., immediate vesting, cliff vesting after 3 years, or graded vesting).
  - **Streams** results back
- Handle errors gracefully (invalid PDF, LLM timeout, etc. Try to think about what are the scenerios that can go wrong)

## Frontend (React or any JS framework)

- File upload component with drag-and-drop support
- Real-time display of extraction progress (streaming output)
- Display extracted fields in editable input fields
- Show loading states and error messages appropriately
- Basic styling (TailwindCSS, MUI, or plain CSS)

**Requirements**

### Streaming Requirement

The extraction must stream results as they become available. You can modify this as you see fit as well:

```
// Example SSE stream format
data: {"field": "plan_name", "value": "Acme Corp 401(k) Plan", "status
    ": "extracted"}

data: {"field": "employer_name", "value": "Acme Corporation", "status
    ": "extracted"}

data: {"field": "employee_contribution_limit", "value": "6%", "status
    ": "extracting"}
...
```

## 4 Provided Resources

- **Sample PDF**: A test 401(k) plan document is in the Documents Folder
- **API Key**: Use your own Gemini API key, or we can provide one upon request. But Gemini provides generous free tier on their api key that you can use.

## 5 Evaluation Criteria

Your submission will be evaluated on:

| Criteria | Weight | What We Look For |
|---|---|---|
| **Functionality** | 40% | Does it work? Can we upload a PDF and see extracted data? |
| **Parallel Processing** | 25% | Can it handle multiple pdfs at the same time without queing? Embarrassingly parallel system should work |
| **Speed** | 10% | Can you make the processing time per extraction for a pdf faster? |
| **UI/UX** | 15% | Is the interface intuitive? Loading states? Error messages? |
| **Documentation** | 10% | README with setup instructions. Code comments where needed. |

## 6 Bonus Points

**Bonus Points**

- **Confidence Scores and Nuance**: Show a confidence level for each extracted field and any nuance from the extracted pdfs
- **Page References**: Indicate which page(s) each field was extracted from
- **Retry Logic**: Allow re-extraction of individual fields
- **Prompt Engineering**: Document your prompt design decisions
- **Testing**: Unit tests for the extraction logic
- **Queuing** : What happens if there are many more pdf processing requests submitted than the system can parallelize at once.

## 7 Submission Instructions

1. Create a **private GitHub repository**
2. Include a `README.md` with:
    - Setup instructions (how to run locally)
    - Architecture overview (brief explanation of your design)
    - Any assumptions or trade-offs you made
    - Time spent on each part (optional but helpful)
3. Add `tahmidawal` as a collaborator
4. Email the repo link to `tahmid.awal@plansync.ai` when complete

## 8 Tips for Success

- **Start simple**: Get a basic working version first, then add features
- **Test with real PDFs**: The provided sample is representative of real documents
- **Show your thinking**: Comments explaining *why* you made certain decisions are valuable
- **AI/LLM based tools**: Use any LLM based tools you want to use or feel comfortable using. We are more interested in your thinking process and code quality.
- **Ask questions**: If something is unclear, email me at tahmid.awal@plansync.ai.

---

Good luck! We're excited to see what you build.

Questions? Contact `tahmid.awal@plansync.ai`