Alright, I'll be blunt: what you're describing is basically a fully-featured AI-powered personal tutor. That's not trivial—it's a full-stack, multi-modal AI system. But it's doable if we break it down smartly. Here's how I'd approach designing it step by step, starting small and scaling up.

Step 1: Define the MVP (Minimum Viable Product)

Focus on **one thing working end-to-end**. Right now, you're trying to do too much at once: text, PDF, images, handwritten notes, quizzes, QA. That's a lot of moving parts. Pick one input modality first—say, **text input** (paragraphs or topic names).

MVP features:

- Student inputs a topic or paragraph.
- LLM generates a **simple explanation**.
- Optional: generate **quiz questions** from the paragraph.

Why: You'll know your core pipeline works before handling PDFs or images, which are way harder.

Step 2: Backend design

You need a system that handles:

1. **User inputs** (text, PDF, images).
2. **Preprocessing** (cleaning text, OCR for images, parsing PDFs).
3. **LLM queries** (sending the cleaned content to an LLM).
4. **Postprocessing** (formatting answers, quizzes, explanations).
5. **Storage** (saving notes, PDFs, generated quizzes).

Tech stack suggestion:

- **Backend**: FastAPI (you already know it).
- **LLM**: OpenAI GPT or local models for cheaper/multi-modal options (like LLaMA + LoRA).
- **Vector DB**: For PDF and image QA (e.g., Chroma, Pinecone, Weaviate).
- **Storage**: PostgreSQL or SQLite for metadata, S3/local for files.

Step 3: Handling multiple input types

- **Text**: direct, send to LLM.

- **PDFs**: extract text via PyPDF2 / PDFplumber, chunk it, embed for retrieval.
- **Images (handwritten notes)**: OCR using **Tesseract** or **AWS Textract**, then treat like text.
- **Audio/Video**: skip for MVP.

Once all are converted to text, you can treat them uniformly.

Step 4: Core LLM workflows

1. **Explain paragraph/topic**: LLM prompt → "Explain this to a high school student in simple terms."
2. **Generate quizzes**: LLM prompt → "Generate 5 multiple-choice questions based on this text."
3. **QA from PDFs/images**:
    - Embed content in a vector database.
    - For a query, retrieve relevant chunks.
    - Send those chunks + query to LLM to generate the answer.

Step 5: Frontend

- Keep it simple at first: a single-page app.
- Features:
    - Input area for text / upload for PDF / upload for image.
    - Button for "Explain," "Generate Quiz," or "Ask a Question."
    - Display area for answer / quiz.

Step 6: Scaling & polishing

After MVP works:

- Add multi-modal handling properly (images + handwritten notes).
- Fine-tune prompts for better quizzes/explanations.
- Add user accounts, progress tracking, and saved notes.
- Consider local embeddings + LLM for offline or cost-efficient solution.

If you want, I can **sketch the full system flow diagram** showing exactly how text, PDFs, and images move through the system to LLMs and come out as explanations or quizzes. It will make the design crystal clear and actionable.

Do you want me to do that next?