**Architectural Plan & Implementation Steps**

Your project can be broken down into five major phases. Your existing files (analysis.py, scoring.py, reporting.py, main.py, utils.py) provide a solid foundation for several of these.

**Phase 1: Knowledge Extraction from PDF**

This phase is the foundation. The goal is to distill the core concepts, topics, and question patterns from the user's uploaded PDF. The analysis.py file is the perfect place for this logic.

**Implementation Steps:**

1. **PDF Parsing (analysis.py):** Use a Python library like PyMuPDF or pdfplumber to extract clean text from the uploaded PDF. This should be a function in utils.py called by analysis.py.
2. **Core Topic Identification (analysis.py):**
   * Feed the extracted text to the Gemini API.
   * **Prompt Engineering is Key:** Your prompt should ask the model to act as an expert educator and identify the 5-7 core knowledge domains or key topic areas covered in the document.
   * **Example Prompt:** "Analyze the following text from an exam paper. Identify and list the top 5 primary subject areas a student must master to succeed. For each area, provide a brief one-sentence description. Format the output as a JSON object."
3. **"Know Me" Questionnaire Generation (analysis.py):**
   * Using the identified topics, instruct Gemini to generate a targeted questionnaire. This bridges the gap between the academic content and the user's life.
   * **Example Prompt:** "For the topic 'Network Security Principles', generate three questions to understand a user's personal context. The questions should be open-ended and relatable. Example: 'Describe a time you've had to secure a personal account or device online.'"

**Phase 2: User Interaction & Scenario Generation**

This phase involves interacting with the user and dynamically creating the scenario-based questions. Your main.py will orchestrate this workflow.

**Implementation Steps:**

1. **Present "Know Me" Questionnaire (main.py):** Display the questions generated in Phase 1 to the user.
2. **Capture User Context (main.py):** Store the user's answers. These personal anecdotes are the raw material for the next step.
3. **Dynamic Scenario Generation (main.py / analysis.py):**
   * This is a crucial step. For each topic from the PDF, combine it with the user's related personal answer and send it to the Gemini API.
   * **Example Prompt:** "Based on the academic topic 'Azure Network Security Groups (NSGs)' and the user's personal experience of 'securing their home Wi-Fi network', create a realistic, scenario-based exam question. The question should require the user to apply the principles of NSGs to their personal experience."
   * **Resulting Question:** "You mentioned securing your home Wi-Fi by blocking unknown devices. Drawing on that experience, explain how you would configure an Azure Network Security Group (NSG) to ensure that a virtual machine hosting a web server only accepts traffic from known IP addresses on port 443, effectively blocking all other access."

**Phase 3: Real-Time Scoring & Multi-Modal Input**

This is where the user answers the scenario questions, and you provide real-time feedback. scoring.py and Eleven Labs integration come into play here.

**Implementation Steps:**

1. **Multi-Modal Input (main.py):**
   * Integrate the Eleven Labs API for voice-to-text transcription.
   * Your application front-end will capture the audio, send it to Eleven Labs, and receive the transcribed text.
2. **Define Scoring Rubric (scoring.py):** For each generated scenario question, make a call to the Gemini API to create a dynamic scoring rubric *before* the user answers.
   * **Example Prompt:** "For the question: '[Insert Scenario Question]', create a JSON scoring rubric. It should include 3-5 key concepts the user must mention to get a full score. Assign points to each concept."
3. **Real-Time Answer Analysis (scoring.py):**
   * As the user types or speaks (and text is transcribed), send the partial or complete answer to your backend.
   * A function in scoring.py will compare the user's answer against the rubric's key concepts. This can be a simple keyword check for the "password-style" feedback (e.g., "✓ *Mentioned NSGs*") or a more advanced semantic check using another Gemini call.

**Phase 4: Predictive Performance Reporting**

After the session, you'll provide the user with their predictive report using reporting.py.

**Implementation Steps:**

1. **Aggregate Scores (reporting.py):** Collect the scores from all answered questions, categorized by the core topics identified in Phase 1.
2. **Generate Insights (reporting.py):**
   * Identify the topics where the user scored lowest. These are their areas for improvement.
   * Use a final Gemini API call to generate a narrative report.
   * **Example Prompt:** "A user scored as follows on these topics: [Insert Topics and Scores]. Generate a predictive performance report. Start with an estimated exam score percentage. Then, list the top 3 areas for improvement with specific, actionable advice for each."

**Phase 5: Orchestration and Utilities**

* **main.py:** This file will act as the controller, orchestrating the entire process from file upload to final report generation by calling functions from the other modules.
* **utils.py:** This file should contain all your helper functions, such as the function to call the Gemini API, the function to call the Eleven Labs API, and any file handling or data cleaning logic. This keeps your main logic clean and reusable.

By following this architectural plan, you can systematically build your Sensa AI feature, leveraging your existing code structure while integrating the power of generative AI for a truly personalized and effective learning experience.