Karan – Task 1: Multimodal Knowledge Ingestion + Agent Response Module (v1)

Checklist:

- Document (PDF/DOCX) upload & parsing via PyMuPDF
- Image upload with OCR (JS/Python bridge using Tesseract or similar)
- Extracted text stored and indexed in MongoDB
- LangChain agent processes uploaded data
- User selects from LLM options (ChatGPT, LLaMA, Grok, UniGuru)
- Generate responses from selected model
- TTS (Text-to-Speech) reads response aloud (API-based or model-hosted)
- REST API or microservice exposed (Python FastAPI or Node wrapper)
- Supabase Auth integrated (token-based access)
- Integration-ready for UniGuru backend (NodeJS) and frontend (Flutter)

Objective:

Build the first version of the Multimodal Ingestion + Response Engine, allowing a user to:

1. Upload documents (PDF/DOCX) and/or images.
2. Extract and index text content using OCR/PyMuPDF.
3. Query the document via an AI agent powered by their selected LLM.
4. Hear the AI agent respond using a TTS engine.
5. Wrap all this in a clean, testable API for later integration.

Steps:

1. File Upload + Parsing:

- Use PyMuPDF to parse PDF/DOCX documents uploaded by the user.
- Set up a lightweight frontend (React or test CLI) to simulate the upload.
- Structure parsed output (title, body, sections) and store in MongoDB.

2. Image to Text (OCR):

- Set up OCR pipeline using Tesseract or EasyOCR (Python preferred).
- Accept image input (JPG/PNG), extract text, append to same MongoDB user document.

3. LangChain Agent Setup:

- Use LangChain to create a basic agent that can answer questions based on the indexed text.
- Agent must pull context from MongoDB.
- Test response generation with a placeholder LLM (like OpenAI or local Ollama).

4. LLM Selector (Drop-down Ready):

- Architect the system to support pluggable LLMs (e.g., ChatGPT, Grok, LLaMA, UniGuru).
- Use environment variables or Mongo config for user-specific model selection.
- Wrap each LLM as a callable function (LangChain or native API).

5. Text-to-Speech:

- Integrate ElevenLabs / Google TTS / custom hosted model.
- Allow agent responses to be read aloud (streamed or downloaded).
- Store/generated audio clips optionally in Supabase Storage.

6. API Wrapping:

- Wrap the above functionality in FastAPI (or NodeJS-compatible service).
- Define clear endpoints:
    - /upload-doc
    - /upload-image
    - /query-agent
    - /get-audio

- Return responses as JSON, audio, or text based on the request.

7. Auth + Integration Hooks:

- Validate user via Supabase Auth JWT before any action.
- Use MongoDB user ID for document linking.
- Ensure endpoints are modular for UniGuru Flutter + NodeJS consumption.

Deliverables:

- GitHub repo with:
  - Code (modular, documented)
  - README with setup and usage guide
  - Postman collection or curl docs for API testing
- Demo: Local test of upload → query → voice response
- Notes for integration with UniGuru app (Flutter frontend & Node backend)

Best of Luck! Please check in with me if you have any doubts.