**Multi-Agentic RAG System**

**Objective**

Design a comprehensive plan for building a Multi-Agentic Retrieval-Augmented Generation (RAG) system tailored for academic environments. This system is not just about answering questions — it's about delivering personalized, syllabus-aligned, and exam-focused responses that evolve based on student feedback and preferences.

**System Architecture Overview**

* Frontend/User Side: Accepts user questions and written feedback. Delivers personalized, syllabus-aligned answers by leveraging retrieval and generation.
* Backend/Admin Side: Allows administrators to input academic data (course content, syllabus, outcomes, etc.) and manages ingestion, structuring, and storage of this data for later use.

**Agents and their Roles**

1. **CourseDataAgent**

* Role: Responsible for loading academic data like syllabus, outcomes, previous year questions, textbooks, and mapping relations between outcomes and topics.
* Functionality: Parses PDFs, Word docs, CSVs, and text files; extracts key fields from structured formats; maps course to outcomes.
* Tools:

PDF/Text/CSV parsers (PyMuPDF, pandas)

SQLite3 writer for structured data (course metadata)

ChromaDB uploader for unstructured text chunks

1. **IndexerAgent**

* Role: Converts unstructured content (syllabus text, textbook paragraphs, questions) into semantically searchable units. This agent prepares the content for efficient retrieval by the question-answering agent by creating vector representations and storing them in a vector database (e.g., ChromaDB).
* Functionality: Splits large texts into manageable chunks and generates dense vector representations.
* Tools:

Chunker (Langchain or custom logic)

Embedding model (Gemini API or SentenceTransformers)

ChromaDB interface for storing chunks

1. **QueryHandlerAgent**

* Role: First responder to any user query, responsible for retrieval and RAG pipeline initiation.
* Functionality: Retrieves relevant document chunks from ChromaDB using semantic similarity and forwards them to the LLM for answer generation.
* Tools:

ChromaDB retriever API

Gemini LLM

1. **PersonalizationAgent**

* Role: Customizes the answer to suit individual user preferences based on historical data and feedback.
* Functionality: Adapts response style (e.g., casual, academic), tone, verbosity, or even preferred learning styles.
* Tools:

User profile manager

Prompt rewriter

1. **CoordinatorAgent**

* Role: Oversees orchestration of all other agents in either parallel or sequential flow.
* Functionality: Coordinates agent tasks using async messaging or centralized logic.
* Tools:

RoundRobinGroupChat / SelectorGroupChat (Langchain or Gemini agents framework**)**

**Data Flow and Interaction**

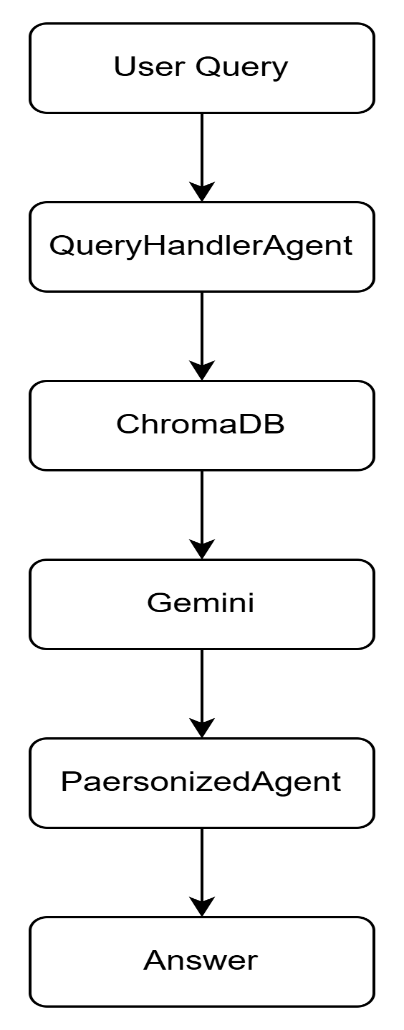
* **Backend Workflow:**

1. Admin uploads academic material: Course name, code, outcomes, previous questions, etc
2. CourseDataAgent: Parses and organizes content. Structured data → SQLite3. Unstructured data → passed to IndexerAgent.
3. IndexerAgent: Chunks and embeds the unstructured text, storing it in ChromaDB



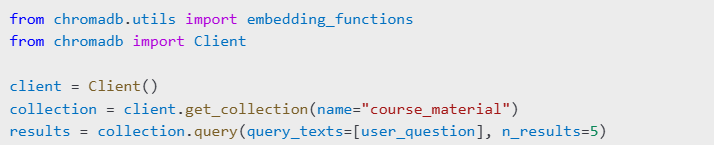
* **Frontend Workflow:**

1. User submits a question → QueryHandlerAgent uses ChromaDB to fetch top-k relevant text chunks.
2. RAG model (Gemini or equivalent) synthesizes the answer based on context retrieved.
3. PersonalizationAgent modifies the answer based on tone, language complexity, or user style.
4. Answer is returned to user.

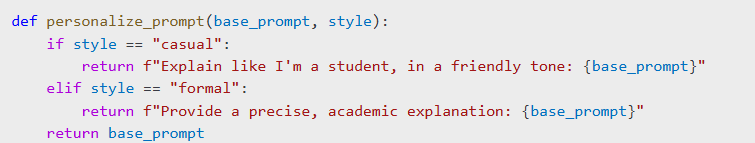


**Example Prompts & Code Sketches**

ChromaDB Retrieval Example:



Personalized Prompt Modification:



**Summary**

This system decomposes academic RAG into modular agents handling ingestion, indexing, query response, personalization, and feedback analysis. Tools like ChromaDB, SQLite3, and Gemini are integrated via these agents. Async orchestration ensures scalability and modular design. Each agent specializes in a narrow task to promote modularity, reusability, and traceability.