**Overview**

This application is an intelligent system designed to handle **document ingestion**, **data retrieval**, and **audio transcription** for applications that require processing and interacting with large datasets. It integrates advanced natural language processing (NLP) techniques and vector-based information retrieval, enabling efficient querying and contextual responses based on ingested data.

The system provides modular APIs to ingest documents, manage session-specific histories, and process audio inputs for transcription, making it a robust solution for knowledge-driven applications.

**Key Features and Capabilities**

**1. Document Ingestion**

The application allows seamless ingestion and indexing of various document formats, enabling efficient retrieval and contextual interaction with the content.

* **Supported Inputs**:
  + **S3 URIs**: Documents stored in Amazon S3 buckets.
  + **Web URLs**: Direct ingestion of documents from HTTPS links.
* **Processing Workflow**:
  + Extracts **text**, **images**, and **metadata** from the provided documents.
  + Stores indexed data in a vector-based database for fast retrieval.
* **Use Case**:
  + Quickly analyze and access relevant content from a large repository of documents without manually sifting through them.

**2. Intelligent Data Retrieval**

The system utilizes a retrieval-based approach to fetch relevant information from indexed documents.

* **How it Works**:
  + A query is processed and matched with the most relevant document chunks using a vector search mechanism.
  + Results are contextualized for clarity and completeness.
* **Customization**:
  + Users can specify which document sets to include in their search.
  + Supports fine-grained filtering to narrow down results.

**3. Audio Processing**

The application supports real-time audio transcription, providing an additional layer of accessibility for users dealing with verbal data.

* **Real-Time Recording**:
  + APIs allow users to start and stop recordings dynamically.
  + Audio data is processed and stored in memory for transcription.
* **Audio File Processing**:
  + Users can upload audio files for immediate transcription.
* **Transcription Model**:
  + Leverages OpenAI's **Whisper** for highly accurate speech-to-text conversion.
* **Output**:
  + Transcriptions are integrated into the system for further processing and querying.

**4. Context Management**

The system supports maintaining and managing context for enhanced query handling:

* **Session History**:
  + Tracks the sequence of user queries and system responses.
  + Allows rephrasing and contextualization of queries for accuracy.
* **Session Management**:
  + Users can clear history for a fresh session.
  + Contextual interactions ensure that each query is understood in relation to its preceding context.

**Detailed API Documentation**

This section provides a comprehensive overview of the APIs available in the application, covering document ingestion, retrieval, and audio processing capabilities. Note that some functionality, such as audio recording, requires client-side implementations to handle hardware interactions like accessing the microphone.

**1. Document Ingestion APIs**

**1.1 /ingest-docs/**

* **Method**: POST
* **Description**: Ingests documents from an S3 URI and indexes them for retrieval.
* **Input**: JSON payload with the following fields:
* {
* "s3\_uri": "string" // URI of the S3 bucket to ingest documents from
* }
* **Output**:
  + Success:
  + {
  + "message": "Successfully processed 5 documents."
  + }
  + Failure: HTTP error code and a message if the URI is invalid or empty.

**1.2 /ingest-urls/**

* **Method**: POST
* **Description**: Ingests documents from a list of HTTPS URLs and indexes for retrieval.
* **Input**: A JSON array of URLs:
* [
* "https://example.com/doc1",
* "https://example.com/doc2"
* ]
* **Output**:
  + Success:
  + {
  + "message": "Documents from URLs successfully ingested."
  + }
  + Failure: HTTP error code and a description of the issue (e.g., invalid URLs).

**2. Retrieval and Query APIs**

**2.1 /ask/**

* **Method**: POST
* **Description**: Handles user queries, retrieves relevant document chunks using multi-step retrieval, and returns a detailed response.
* **Input**: JSON payload with the following fields:
* {
* "query": "string", // The user query
* "session\_id": "string", // Session ID for tracking context
* "doc\_list": ["string"] // List of document names for filtering
* }
* **Output**:
  + Success:
  + {
  + "chat\_history": [
  + ["human", "What are the key features?"],
  + ["ai", "The key features are ..."]
  + ],
  + "session\_id": "session\_123",
  + "image\_s3": ["s3://bucket/image1", "s3://bucket/image2"]
  + }
  + Failure: HTTP error code with error details.

**2.2 /clear-chat/**

* **Method**: POST
* **Description**: Clears the chat history for a given session.
* **Input**: JSON payload with the session ID:
* {
* "session\_id": "string"
* }
* **Output**:
* {
* "message": "Chat history cleared."
* }

**3. Audio Processing APIs**

**3.1 /recording**

* **Method**: POST
* **Description**: Handles audio recording actions (start/stop) for real-time transcription and querying.
* **Input**: JSON payload with the following fields:
* {
* "action": "start|stop", // Action to perform: start or stop recording
* "session\_id": "string", // Session ID for context
* "doc\_list": ["string"] // List of documents for filtering during query resolution
* }
* **Output**:
  + On start:
  + {
  + "message": "Recording started, press Enter to stop it."
  + }
  + On stop:
  + {
  + "chat\_history": [
  + ["human", "What did the user say?"],
  + ["ai", "Here is the response."]
  + ],
  + "session\_id": "session\_456",
  + "image\_s3": ["s3://bucket/image1"]
  + }
* **Important Note**:
  + This API only handles the backend processing logic.
  + **Microphone Access**: Audio recording requires a client-side implementation to access the microphone. For example:
    - In a web application, use JavaScript APIs like getUserMedia.
    - For desktop or mobile apps, use platform-specific libraries to capture audio data.
  + The client-side implementation must send the recorded audio to the backend for processing.

**3.2 /voice\_assistant**

* **Method**: POST
* **Description**: Processes uploaded audio files, transcribes the content, and retrieves document-based responses.
* **Input**: Multipart form data with the following fields:
  + audiofile (binary): The uploaded audio file.
  + session\_id (string): Session ID for tracking.
  + doc\_list (array of strings): List of documents for filtering.
* **Output**:
  + Success:
  + {
  + "chat\_history": [
  + ["human", "Audio query transcription."],
  + ["ai", "Response based on transcription."]
  + ],
  + "session\_id": "session\_789",
  + "image\_s3": ["s3://bucket/image1"]
  + }
  + Failure: HTTP error code with error details.

**4. Multi-Step Retrieval Mechanism**

The application incorporates a multi-step retrieval process to maximize the relevance and accuracy of responses:

1. **Query Reformulation**:
   * Reformulates user queries based on the session’s chat history to ensure clarity and completeness.
2. **Initial Retrieval**:
   * Retrieves a set of relevant document chunks from the vector database.
3. **Contextual Refinement**:
   * Iteratively refines results by rerunning queries with additional filters or context.
4. **Response Generation**:
   * Synthesizes the most relevant information into a final response.

**5. Key Considerations**

* **Session Management**:
  + Every interaction is tied to a session for tracking and context-building.
  + Sessions can be reset using the /clear-chat/ API.