Technical Documentation: Simple RAG-Based Document Ingestion System

Overview

The **Simple RAG-Based Document Ingestion System** enables the upload and processing of documents using **Retrieval-Augmented Generation (RAG)**. The system leverages **Supabase** for database management, **OpenAI embeddings** for document vectorization, and **LangChain** for text splitting. This setup facilitates seamless document ingestion and retrieval, allowing for the extraction and querying of information from uploaded documents.

Features

1. **Document Upload**
    - Users can upload PDF documents along with a title and description.
2. **Document Ingestion**
    - Documents are ingested into a vector database, breaking them down into chunks and generating embeddings to make the document content searchable.
3. **Progress Tracking**
    - The ingestion process is tracked with status updates and progress percentages, allowing users to monitor the process (0% → 100%).

4. **APIs for Management**
    - APIs are provided to manage the document upload and ingestion process, including progress tracking and retrieval.
5. **RAG-Based Retrieval**
    - Using RAG, the system allows for efficient querying and information retrieval based on previously ingested documents.

Tech Stack

Backend

- **Next.js**: Utilized for API routes to handle document uploads and ingestion.
- **Formidable**: Used for handling file uploads efficiently.
- **Supabase**: Acts as the database, storage, and vector store for ingested documents.
- **LangChain**: Manages text splitting and setup for RAG, allowing the document to be split into smaller, meaningful chunks.
- **OpenAI Embeddings**: Employed to generate document embeddings, making it possible to retrieve relevant information based on text queries.

How It Works

1. Upload API

- **Objective**: Allows users to upload PDF documents along with title and description metadata.

- **Flow**: The document and metadata are stored in Supabase, while the document file path is saved for further processing.

2. Ingestion API

- **Objective**: Processes the uploaded PDF document by:
    - Extracting text from the PDF.
    - Splitting the text into smaller, manageable chunks.
    - Generating embeddings for each chunk using OpenAI embeddings.
    - Storing the vectors and associated metadata in Supabase.
- **Ingestion Progress Tracking**:
    - The ingestion status is updated with the following parameters:
        - `status`: Current status (processing / completed).
        - `progress`: Percentage of completion (0 to 100).
        - `startedAt`: Timestamp indicating when ingestion started.

    - `completedAt`: Timestamp indicating when ingestion completed.

3. Update Progress API

- **Objective**: Manually update the progress of a document's ingestion process in real-time.
- **Use Case**: Useful during large file processing or slow ingestion tasks where progress needs to be updated periodically.

4. Get Ingestion Status API

- **Objective**: Retrieve the current status of an ingestion job.
- **Use Case**: To check the status of ongoing or completed ingestion tasks, such as retrieving the percentage progress or the final status.

API Endpoints

API Endpoint	Method	Description
`/api/upload`	POST	Upload a PDF file along with title and description.
`/api/ingestion`	POST	Initiate ingestion for the uploaded document.
`/api/ingestion`	GET	Retrieve documents with their ingestion status (processing/completed).

`/api/ask`	POST	Ask a question, and RAG will retrieve relevant information based on the ingested document.
`/api/get-documents`	GET	Retrieve uploaded documents and their ingestion status.

Setup Instructions

Clone the Repository

```
git clone https://github.com/coolpinkzz/simple-rag-ingestion.git
cd simple-rag-ingestion
```

Install Dependencies

```
npm install
```

Create .env.local Configuration File

Ensure that you have a `.env.local` file with the following environment variables:

```
NEXT_PUBLIC_SUPABASE_URL=your-supabase-url
NEXT_PUBLIC_SUPABASE_SERVICE_ROLE_KEY=your-supabase-service-role-key
OPENAI_API_KEY=your-openai-api-key
```

Run the Development Server

```
npm run dev
```

Database Schema

Table: `uploads`

Field	Type	Description
`id`	UUID (Primary Key)	Unique identifier for the document
`title`	Text	Title of the document
`description`	Text	Description or metadata for the document
`file`	Text	File path or URL where the document is stored

Table: `ingestions`

Field	Type	Description
`id`	UUID (Primary Key)	Unique identifier for the ingestion task
`documentId`	UUID (Foreign Key to `uploads`)	Reference to the document being ingested
`documentTitle`	Text	Title of the document being ingested
`status`	Text	Status of the ingestion (processing / completed)
`progress`	Integer (0 to 100)	Percentage of completion of the ingestion

`startedAt`	Timestamp	Timestamp indicating when ingestion started
`completedAt`	Timestamp	Timestamp indicating when ingestion completed

Conclusion

This **RAG-based document ingestion system** provides a seamless way to upload, process, and retrieve information from large documents. By leveraging powerful tools such as Supabase for storage, OpenAI embeddings for vectorization, and LangChain for efficient text splitting, the system enables high-performance document retrieval.

The ingestion process is tracked in real-time, providing visibility into document processing, and ensuring that users can effectively monitor and query ingested documents as needed.

The passwords are all more than 6 characters long for security. Below is the structure of the mock users:

1. **Admin User:**
    - Username: `admin`
    - Email: `admin@example.com`
    - Role: `admin`
    - Password: `admin123`

- Created At: `2023-01-01T00:00:00Z`

2. **User 1:**
   - Username: `user1`
   - Email: `user1@example.com`
   - Role: `user`
   - Password: `user12345`
   - Created At: `2023-01-02T00:00:00Z`
3. **User 2:**
   - Username: `user2`
   - Email: `user2@example.com`
   - Role: `user`
   - Password: `password789`
   - Created At: `2023-01-03T00:00:00Z`

Please use the above data for your mock users. Ensure that passwords are stored securely in your application (ideally hashed) when working with real data.