TITLE AND OVERVIEW

Title: Document Management and RAG-based Q&A Application

Version: 1

Date: 28/2/2025

Author: Aditya Dogra

INTRODUCTION

Purpose: The purpose of this application is to develop a backend application to handle document ingestion, embedding generation and retrieval based Q&A(RAG).

Scope: The application aims to manage users, documents and an ingestion process that generates embedding for document retrieval in a Q&A setting.

Objectives: Accept document data, generates embedding using a LLM library and stores them for future retrieval. Accepts user questions, retrieve relevant document embeddings and generates answer on the retrieved content using RAG. Enables user to specify which documents to consider in the RAG based Q&A process.

Stakeholder: Python Developer named Aditya is involved.

SYSTEM ARCHITECTURE

High Level Diagram:

1. **Document Ingestion**
   Upload documents→ Generate embeddings→ Store in DB.
2. **Question Answering(RAG)**
   User questions→ Retrieve Relevant embeddings → Use LLM to generate answers.
3. **Document selection**
   Specify documents → Filter retrieval step by selected documents.
4. **Token Generation**
   Create a user with username and password→ Generate an access token to access different routes.

Component Breakdown

1. **Document Ingestion Service**
   Parses documents (e.g., PDF, DOCX, TXT).
   Generates embeddings (via LLM)
   Stores embeddings and metadata in the DB.

2. **Retrieval Service**

   Searches for relevant document chunks using vector similarity (e.g., cosine similarity)

   Returns top-N matching document sections.

3. **Answer Generation Service**
   Takes the retrieved content and user query.
   Uses an LLM to generate an answer via a RAG pipeline.
4. **Document Management Service**
   Allows users to select, list, or delete documents.
5. **Token Generation Service:**
   A service to generate a unique token to access different routes.

TECHNOLOGY STACK

1. **Backend (FastAPI)**: Handles API requests, ingestion, and retrieval.
2. **Vector Database (Postgres + pgvector)**: Stores document embeddings.
3. **LLM (Hugging Face)**: Generates embeddings and answers user queries.
4. **File Storage** : For raw documents on PostgreSQL

API DESIGN

1. Document Ingestion API:

POST /ingest

Also specify the bearer token generated from /token in the request

Request Body:

```
{
    "title": "Sample Doc7",
    "content": "BJP won this time election in 2025 by beating AAP. They got 48 votes this time as compared to 22 received by AAP. AAP has to pay the price for building their sheesh mahal and the liqour scam. Rekha Gupta is announced as the new CM of delhi with pravesh verma being the PWD minister "
}
```

Response:

```
{
    "message": "Document ingestion started in the background",
    "document": {
        "id": null,
        "metadata": {
            "title": "Sample Doc7"
        },
        "page_content": "BJP won this time election in 2025 by beating AAP. They got 48 votes this time as compared to 22 received by AAP. AAP has to pay the price for building their sheesh mahal and the liqour scam. Rekha Gupta is announced as the new CM of delhi with pravesh verma being the PWD minister ",
        "type": "Document"
    }
}
```

2. Q & A API

POST /qa

Also, specify the bearer token generated from /token in the request.

Request Body:

```
{
    "question": "Who is the new CM of delhi?",
    "top_k": 3
}
```

Response:

Rekha Gupta(Streaming Response)

3. **Document Selection API**
   GET /documents

   Also, specify the bearer token as generated from '/token' in the request.

   Response:


```
[
    {
        "title": "Test Doc",
        "content": "Test content."
    },
    {
        "title": "Test Doc",
        "content": "Test content."
    },
    {
        "title": "Test Doc",
        "content": "Test content."
    },
```


4. **Token Generation API**

POST /token

Request Body:

curl --location 'http://localhost:8000/token' \

--header 'Content-Type: application/x-www-form-urlencoded' \

--data-urlencode 'username=aditya' \

--data-urlencode 'password=1234'

Here, the header, username and password is specified to the generate the access token.

Response:


```
{
    "access_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZGl0eWEiLCJleHAiOjE3NDA5MjQwODd9.
4BO850bPoMPIevHuX5LRvWN47TumJ9E9gmoL1sa-C9o",
    "token_type": "bearer"
}
```

DATABASE DESIGN

Documents Table

```
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255),
    content TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Embedding Table

```
CREATE TABLE embeddings (
    id SERIAL PRIMARY KEY,
    document_id INT REFERENCES documents(id),
    embedding VECTOR(1536), -- or appropriate size for your model
);
```

Users Table

```
CREATE TABLE Users(
   Id SERIAL PRIMARY KEY,
   username VARCHAR(255),
   hashed_password VARCHAR(255),
   is_active BOOLEAN
);
```

ERROR HANDLING AND LOGGING

1. To get the current user, if the token entered is invalid then ‘Invalid Token’ message is raised.

SECURITY CONSIDERATION

1. **Authentication:**

- **OAuth2PasswordBearer (oauth2_scheme)**

  Defines the token URL (/token) for obtaining access tokens.

  FastAPI's Depends function automatically extracts the token from the Authorization header.

- **JWT Token Creation (create_access_token)**

  Encodes user data with an expiration time.

  Uses a secret key (SECRET_KEY) and algorithm (ALGORITHM) to sign the token.

- **Token Verification (get_current_user)**

  Decodes the token to extract the username.

  Queries the database for the user.

  Raises an error if the token is invalid or the user is not found.

2. **Authentication Flow**

- **Login Request:**

  User sends credentials (username, password) to /token.

- **Validation:**

  System queries the database for the user.

  Validates the password using verify_password.

- **Token Generation:**

  If credentials are correct, the system generates a JWT with the username as the subject (sub) and an expiration time.

- **Accessing Protected Routes:**

  User includes the token in the Authorization header as a **Bearer token**.

  FastAPI extracts the token and verifies it using get_current_user.

- **Token Validation:**

  System decodes the JWT using the secret key and algorithm.

  Validates the token expiration.

  Fetches the user from the database.

  Raises an HTTPException for invalid credentials or expired tokens.

FUTURE ENHANCEMENTS

- Accuracy of the results could be improved with higher end GPUs or available paid models with API keys like Llama, GPT 4-o etc.
- For very large document that is above the context length of the models, code could be optimized with chunking strategies.
- Fine tuning can be done for specific domains.
- Support for document types can be added like Excel, HTML etc.

APPENDIX

1. **Glossary**
    - **RAG:** Retrieval Augmented Generation
    - **LLM:** Large Language Model
2. **References**
    - https://www.langchain.com/
    - https://supabase.com/docs/guides/database/extensions/pgvector
    - https://fastapi.tiangolo.com/