FastAPI-Based Document Processing and Query System

Overview

This project is a FastAPI-based application designed to process and query documents. It includes features such as user authentication, document ingestion, and querying using embeddings and a language model. The application is containerized using Docker and supports deployment via Azure Pipelines.

Features

1. **User Authentication**:
    - User registration and login with hashed passwords.
    - JWT-based authentication with access and refresh tokens.
    - Token refresh functionality.
2. **Document Ingestion**:
    - Upload and process PDF documents.
    - Extract text from PDFs and generate embeddings using a pre-trained model.
3. **Document Querying**:
    - Query documents using embeddings to find the most relevant content.
    - Use a language model to generate answers based on the queried content.
4. **Admin Features**:
    - List all ingested documents.
5. **Deployment**:
    - Dockerized application for easy containerization.
    - CI/CD pipeline using Azure Pipelines for testing, building, and deploying the application.

Project Structure

```
backend/

├── alembic/                # Database migrations

├── models/                 # Database models

├── postgres/               # Database connection setup

├── routers/                # API routes

├── schemas/                # Pydantic schemas for request/response
validation
```

```
├── utils/                    # Utility modules (e.g., logging, OpenAI integration)
├── main.py                   # Entry point for the FastAPI application
├── requirements.txt          # Application dependencies
├── test_requirements.txt     # Testing dependencies
├── test_main.py              # Unit tests
├── Dockerfile                # Docker configuration
├── azure-pipelines.yml       # Azure Pipelines configuration
└── alembic.ini               # Alembic configuration
```

Installation

Prerequisites

- Python 3.10 or higher
- PostgreSQL database
- Docker (optional, for containerized deployment)

Running the Project Locally

Steps

1. **Clone the Repository**:
2. `git clone <repository-url>`
3. `cd backend`
4. **Set Up Environment Variables**: Create a `.env` file in the root directory and add the following variables:
5. `DATABASE_URL=postgresql+asyncpg://<username>:<password>@<host>:<port>/<database>`
6. `SECRET_KEY=<your-secret-key>`
7. `ALGORITHM=HS256`
8. **Install Dependencies**: Install the required Python packages:
9. `pip install -r requirements.txt`
10. **Set Up the Database**:
    - Ensure PostgreSQL is running and accessible.
    - Run Alembic migrations to create the database schema:
    - `alembic upgrade head`

11. **Start the Application**: Run the FastAPI application using Uvicorn:
12. `uvicorn main:app --reload`
13. **Access the API**: Open your browser and navigate to `http://localhost:8000/docs` to view the interactive API documentation.

API Endpoints

Authentication

- **POST** `/auth/register`: Register a new user.
- **POST** `/auth/login`: Log in and receive access and refresh tokens.
- **POST** `/auth/token/refresh`: Refresh the access token.
- **GET** `/auth/me`: Get the current user's details.

Document Processing

- **POST** `/api/ingest/`: Upload and process a PDF document.
- **POST** `/api/query/`: Query documents using a question.
- **GET** `/api/documents/`: List all ingested documents.

Root

- **GET** `/`: Welcome message.

Configuration

Environment Variables

- `DATABASE_URL`: PostgreSQL connection string.
- `SECRET_KEY`: Secret key for JWT token generation.
- `ALGORITHM`: Algorithm for JWT encoding/decoding.

Logging

Logging is configured in logger.py. Logs are output to the console.

Testing

Running Tests

1. Install test dependencies:
2. `pip install -r test_requirements.txt`
3. Run tests using `pytest`:
4. `pytest`

Deployment

Docker

1. Build the Docker image:
2. `docker build -t fastapi-app .`
3. Run the container:
4. `docker run -p 8000:8000 fastapi-app`

Azure Pipelines

The CI/CD pipeline is defined in azure-pipelines.yml. It includes the following stages:

1. **Test**: Run unit tests.
2. **Build**: Build and push the Docker image.
3. **Deploy**: Deploy the application to different environments (Dev, UAT, Staging, Prod).

Key Modules

endpoints.py

Handles document ingestion and querying:

- Extracts text from PDFs.
- Generates embeddings using `sentence-transformers/all-MiniLM-L6-v2`.
- Queries documents based on embeddings and retrieves answers using OpenAI's GPT model.

models

Defines database models for users, tokens, and documents.

schemas

Defines Pydantic schemas for request and response validation.

openai.py

Integrates with OpenAI's GPT model for generating answers.
