VectorDB/db.py

This script ingests PDF documents into a Weaviate vector database. It extracts text, encodes the PDF as a BLOB, and stores both in a new collection.

Imports

- `os`, `base64` for file handling and encoding
- `logging` for event reporting
- `weaviate` client libraries for DB operations
- `pypdf` for PDF text extraction

```
import os
import logging
import weaviate
from weaviate.classes.init import Auth
from weaviate.classes.config import Configure, Property, DataType
import pypdf
import base64
```

Setup & Logging

- Configures `logging` at INFO level
- Establishes a logger named **weaviate-demo**
- Logs essential events and errors

Weaviate Connection

- Reads `WEAVIATE_URL` and `WEAVIATE_API_KEY`
- Exits on missing credentials
- Connects via `weaviate.connect_to_weaviate_cloud` and logs success/failure

Collection Management

- Defines `COLLECTION_NAME = "PythonDocs"`
- Checks if the collection exists; deletes it for a fresh start
- Creates a new collection with:
  - **text** (TEXT)
  - **file** (BLOB)
  - **filename** (TEXT)
- Configures embeddings using Snowflake's `text2vec_weaviate` model

PDF Ingestion

- Verifies the presence of `example.pdf`
- Reads PDF bytes and extracts text with `pypdf.PdfReader`
- Encodes PDF bytes to Base64 for BLOB storage
- Inserts properties `{file, filename, text}` into the collection
- Closes the Weaviate client upon completion

Ingestion Pipeline Diagram

```
flowchart TD
    A[Start] --> B[Setup Logging]
    B --> C[Connect to Weaviate]
    C --> D[Reset/Create Collection]
    D --> E[Read PDF file]
    E --> F[Extract text]
    F --> G[Encode PDF to Base64]
    G --> H[Insert into Collection]
    H --> I[Close Client]
    I --> Z[End]
```

Relationships

- Depends on packages in **backend/requirements.txt**
- Populates data consumed by the RAG API in **backend/rag.py**

---

backend/Dockerfile

This Dockerfile containers the RAG API for production. It installs system and Python dependencies, configures a non-root user, and defines the runtime.

Base Image & Environment

- **Base**: `python:3.12-slim` for minimal footprint
- Sets `PYTHONDONTWRITEBYTECODE`, `PYTHONUNBUFFERED`, `PYTHONPATH=/app`, `PORT=8000`

Dependency Installation

- Installs system packages: `build-essential`, `curl`
- Cleans `apt` lists to reduce layer size
- Adds `appuser` non-root group and user
- Copies `requirements.txt` for Docker cache optimization

- Upgrades `pip` and installs Python dependencies

Application Setup

- Creates `/app/logs` and grants ownership to `appuser`
- Copies application code into `/app`
- Recursively sets `/app` ownership to `appuser`
- Switches to `appuser` context for security

Runtime Configuration

- Exposes port `8000`
- Defines a health check probing `/healthcheck` every 30s
- Runs the app via Uvicorn:

```
python -m uvicorn rag:app --host 0.0.0.0 --port 8000 --workers 1
```

Build & Run Flowchart

```
flowchart LR
    A[Code Checkout] --> B[Docker Build]
    B --> C[Install System Deps]
    C --> D[Install Python Deps]
    D --> E[Copy Code & Logs]
    E --> F[Set User & Ports]
    F --> G[Define Healthcheck]
    G --> H[Start Uvicorn]
```

Relationships

- Reads dependencies from **backend/requirements.txt**
- Launches the FastAPI app defined in **backend/rag.py**

---

backend/rag.py

This FastAPI application implements a production-ready RAG (Retrieval-Augmented Generation) service. It uses Weaviate and Cohere integrations to answer prompts with source citations.

Logging Configuration

- Creates `logs/` directory on startup
- Defines **detailed** and **simple** formatters

- Attaches: console handler, full-file handler (`logs/app_YYYYMMDD.log`), error-file handler (`logs/errors_YYYYMMDD.log`)
- Uses a root logger at INFO level

Configuration Class

```
class Config:
    WEAVIATE_URL: str
    WEAVIATE_API_KEY: str
    COHERE_API_KEY: str
    COLLECTION_NAME: str = "PythonDocs"
    API_TITLE: str
    API_DESCRIPTION: str
    API_VERSION: str
    ALLOWED_ORIGINS: List[str]
    TRUSTED_HOSTS: List[str]
    DEFAULT_LIMIT: int = 3
    DEFAULT_ALPHA: float = 0.5
    MAX_CONTENT_LENGTH: int = 1000
```

- Loads from environment or defaults for dev and prod

Weaviate Client Initialization

- Uses `weaviate.use_async_with_weaviate_cloud` with Cohere headers
- Logs client creation success or error
- Provides `async_client` for all DB calls

Data Models

- **PromptRequest**: input prompt with length limits
- **SearchResult**: individual item with `source` , `answer` , `confidence`
- **AnswerResponse**: collection of results, timing, and count
- **HealthResponse**: status, timestamp, Weaviate readiness, version
- **ErrorResponse**: standardized error payload

Utility Functions

- `check_collection_exists(name)` : returns bool if Weaviate collection exists
- `list_available_collections()` : lists all Weaviate collection names

Middleware & Lifespan

- **Lifespan**: connects/disconnects `async_client` on startup/shutdown

- **CORS**: allows origins from config
- **TrustedHost**: restricts host headers
- **Request Logging**: logs each request and response time

Exception Handling

- Catches `HTTPException` and returns `ErrorResponse` with status code
- Catches generic exceptions, logs trace, and returns 500 with `ErrorResponse`

API Endpoints

- `GET /` → API info (message, version, docs, health URLs)
- `GET /healthcheck` → detailed health status
- `GET /collections` → lists collections & configured existence
- `GET /debug/collection` → debug info for target collection
- `GET /metrics` → basic API metrics
- `POST /answer` → main RAG endpoint returning `AnswerResponse`

Sequence Diagram: `/answer` Flow

```
sequenceDiagram
    participant C as Client
    participant A as API Server
    participant W as Weaviate
    C->>A: POST /answer { prompt }
    A->>W: is_ready()
    W-->>A: ready
    A->>W: generate.hybrid(query, limit, alpha)
    W-->>A: objects
    alt generative success
        A->>A: build SearchResult list
    else fallback
        A->>W: query.hybrid(...)
        W-->>A: objects
        A->>A: build SearchResult list
    end
    A-->>C: AnswerResponse(results)
```

Relationships

- Uses the Weaviate data populated via **VectorDB/db.py**
- Depends on Python packages listed in **backend/requirements.txt**

---

backend/requirements.txt

This file lists all Python dependencies for both ingestion and API layers.

```
weaviate-client>=4.9.5
pypdf
python-dotenv
fastapi
uvicorn
pydantic
cohere
gunicorn
python-multipart
```

Dependency Overview

Package	Purpose
weaviate-client>=4.9.5	Connect and interact with Weaviate
pypdf	Extract text from PDF documents
python-dotenv	Load environment variables
fastapi	Build the RESTful API
uvicorn	ASGI server for FastAPI
pydantic	Data validation and settings management
cohere	Integrate Cohere generative models
gunicorn	WSGI server for production deployment
python-multipart	Parse `multipart/form-data` requests

Usage

- Installed during Docker build
- Required for both **db.py** and **rag.py** scripts to run properly

Relationships

- Consumed by **backend/Dockerfile** to install dependencies

- Powers both PDF ingestion and RAG API services