📄 RAG Ingestion Pipeline Documentation

Overview

This system enables users to upload one or more PDFs via their URLs and generate vector embeddings using OpenAI Embeddings. These embeddings are chunked and stored in a user-specific **Qdrant collection** for retrieval during future queries.

It is built using **FastAPI** for the web server and integrates the **LangChain**, **Qdrant**, and **OpenAI** libraries for document processing and embedding generation.

📁 File Structure

```
project/
├── rag_api.py        # FastAPI endpoint for PDF ingestion
├── rag_builder.py    # Embedding generation and Qdrant storage logic
├── .env              # Stores environment variables for Qdrant
```

🔧 Environment Variables

Place these in a `.env` file in your root directory:

```
QDRANT_URL=https://your-qdrant-instance.com
QDRANT_API_KEY=your-qdrant-api-key
```

📦 Dependencies

Install these packages (usually via `pip install -r requirements.txt`):

```
fastapi
uvicorn
```

```
pydantic
python-dotenv
requests
langchain
langchain-community
langchain-openai
langchain-qdrant
qdrant-client
```

You can also install them manually via:

```
pip install fastapi uvicorn pydantic python-dotenv requests \
            langchain langchain-community langchain-openai
langchain-qdrant \
            qdrant-client
```

🔄 API Endpoint

`POST /ingest`

Description: Triggers PDF ingestion and embedding.

Request Body:

```
{
  "pdf_urls": ["https://example.com/file1.pdf",
"https://example.com/file2.pdf"],
  "user_id": "user123",
  "reset": true
}
```

Field	Type	Description
pdf_urls	string[]	List of public URLs to PDFs
user_id	string	User identifier to isolate their embeddings in Qdrant

reset	bool	Whether to reset the user's Qdrant collection before ingest

Response:

```
{
  "status": "ok",
  "message": "Embeddings successfully created."
}
```

On error:

```
{
  "status": "error",
  "detail": "Error message here"
}
```

⚙️ How It Works

1. API Layer (`rag_api.py`)

- Exposes a single endpoint `/ingest`.
- Accepts PDF URLs and user ID in the request.
- Calls `rag_builder.main_from_api(...)` in a background thread to avoid blocking.
- Returns success or error response.

2. Embedding Pipeline (`rag_builder.py`)

Key Steps:

1. **PDF Download**:

    - Each PDF URL is downloaded using `requests`.

2. **Document Loading**:
    - Uses `PyPDFLoader` from LangChain to extract text from the downloaded PDFs.
3. **Chunking**:
    - Documents are split into overlapping chunks (300 characters long with 30 character overlap) using `RecursiveCharacterTextSplitter`.
4. **UUID Tagging**:
    - Each chunk is tagged with a UUID (`chunk.metadata["id"]`).
5. **Qdrant Vector DB**:
    - If `reset=True`, deletes and recreates the user's Qdrant collection.
    - Embeds each chunk using OpenAI (`text-embedding-3-small` model).
    - Stores embedded vectors in the specified Qdrant collection.

📌 Notes

- The embeddings are stored in collections named: **`user_{user_id}_embeddings`** (e.g., `user_123_embeddings`).
- If no valid text is found in any PDF, ingestion fails with an appropriate error.
- The app is designed to **work asynchronously**, suitable for deploying in production.

▶️ Running the API

To run the FastAPI app locally:

```
uvicorn rag_api:app --reload
```

Test with `curl`, Postman, or any HTTP client.

✅ Example Use Case

A user uploads 3 PDFs via their URLs. The system:

- Downloads each PDF
- Extracts and chunks the text
- Embeds all chunks
- Stores embeddings in Qdrant under their unique collection
