Complete Setup Guide for AI-Powered PDF Document System

Prerequisites

Before you begin, ensure you have the following installed on your machine:

1. **Python 3.8 or higher**
    - Download from python.org
    - Verify installation: `python --version`
2. **Git**
    - Download from git-scm.com
    - Verify installation: `git --version`
3. **API Keys (Required)**
    - OpenAI API Key (for embeddings and language model)
    - Sign up at platform.openai.com

Step 1: Clone and Setup the Repository

1.1 Clone the Repository

bash

```bash
git clone https://github.com/Ad1ty4shrma/AI-assignment.git
cd AI-assignment
```

1.2 Create a Virtual Environment

bash

```bash
# Create virtual environment
python -m venv venv

# Activate virtual environment
# On Windows:
venv\Scripts\activate
# On macOS/Linux:
source venv/bin/activate
```

1.3 Install Dependencies

bash

```bash
# Install required packages
pip install -r requirements.txt

# If requirements.txt doesn't exist, install manually:
pip install fastapi uvicorn python-multipart
pip install chromadb sentence-transformers
pip install PyMuPDF langchain openai
pip install python-dotenv streamlit
```

Step 2: Environment Configuration

2.1 Create Environment Variables File

Create a `.env` file in the root directory:

bash

```bash
touch .env
```

2.2 Add API Keys to .env File

env

```env
OPENAI_API_KEY=your_openai_api_key_here
CHROMA_DB_PATH=./chroma_db
UPLOAD_FOLDER=./uploads
```

2.3 Create Required Directories

bash

```bash
mkdir uploads
mkdir chroma_db
mkdir static
mkdir templates
```

Step 3: Verify Your Project Structure

Your project should look like this:

```
AI-assignment/
├── .env
├── requirements.txt
├── app.py (main FastAPI application)
├── pdf_processor.py (PDF processing logic)
├── vector_store.py (ChromaDB operations)
├── query_engine.py (RAG implementation)
├── uploads/ (uploaded PDFs)
├── chroma_db/ (vector database)
├── static/ (CSS/JS files)
├── templates/ (HTML templates)
└── README.md
```

Step 4: Running the Application

4.1 Start the FastAPI Server

bash

```bash
# Method 1: Using uvicorn directly
uvicorn app:app --reload --host 0.0.0.0 --port 8000

# Method 2: If using python -m
python -m uvicorn app:app --reload
```

4.2 Alternative: Streamlit Interface (if available)

bash

```bash
streamlit run streamlit_app.py
```

Step 5: Testing the Application

5.1 Access the Web Interface

- Open your browser and go to: `http://localhost:8000`
- For API documentation: `http://localhost:8000/docs`

5.2 Test PDF Upload

1. Navigate to the upload page
2. Select a PDF file from your computer

3. Click "Upload" to process the PDF
4. Wait for processing to complete

5.3 Test Query Functionality

1. After uploading PDFs, go to the query page
2. Enter a natural language question
3. Click "Ask" to get an answer
4. Review the response and source information

Step 6: Common Issues and Solutions

6.1 Installation Issues

bash

```bash
# If you get permission errors on Windows:
pip install --user -r requirements.txt

# If you get SSL certificate errors:
pip install --trusted-host pypi.org --trusted-host pypi.python.org -r requirements.txt

# For M1/M2 Mac users:
pip install --no-deps sentence-transformers
```

6.2 API Key Issues

- Ensure your OpenAI API key is valid and has credits
- Check that the `.env` file is in the correct location
- Verify the environment variable names match your code

6.3 ChromaDB Issues

bash

```bash
# If ChromaDB fails to initialize:
pip install --upgrade chromadb

# Clear existing database:
rm -rf chroma_db/*
```

6.4 PDF Processing Issues

```bash
# If PyMuPDF fails to install:
pip install --upgrade pymupdf

# Alternative PDF libraries:
pip install pdfplumber pdfminer.six
```

Step 7: Key Features and Usage

7.1 PDF Upload and Processing

- Supports multiple PDF formats
- Automatically extracts text and creates embeddings
- Stores in ChromaDB for efficient retrieval

7.2 Query Interface

- Natural language question input
- Context-aware responses using RAG
- Shows source document and page references

7.3 API Endpoints

- `POST /upload` - Upload PDF files
- `POST /query` - Submit questions
- `GET /docs` - API documentation

Step 8: Advanced Configuration

8.1 Customize Embedding Model

Edit your configuration to use different models:

```python
# In your code, you can change:
model_name = "sentence-transformers/all-MiniLM-L6-v2"
# or
model_name = "sentence-transformers/all-mpnet-base-v2"
```

8.2 Adjust Chunk Size

python

```python
# Modify text chunking parameters:
chunk_size = 1000
chunk_overlap = 200
```

8.3 Configure Vector Database

python

```python
# ChromaDB settings:
collection_name = "pdf_documents"
distance_metric = "cosine"
```

Step 9: Production Deployment

9.1 Docker Setup (Optional)

bash

```bash
# Build Docker image:
docker build -t pdf-ai-system .

# Run container:
docker run -p 8000:8000 pdf-ai-system
```

9.2 Environment Variables for Production

env

```env
ENVIRONMENT=production
DEBUG=False
OPENAI_API_KEY=your_production_key
```

Step 10: Troubleshooting Commands

10.1 Check Python Environment

bash

```bash
python --version
pip list
which python
```

10.2 Test Individual Components

bash

```bash
# Test PDF processing:
python -c "import PyMuPDF; print('PDF processing OK')"

# Test ChromaDB:
python -c "import chromadb; print('ChromaDB OK')"

# Test OpenAI:
python -c "import openai; print('OpenAI OK')"
```

10.3 View Logs

bash

```bash
# Run with verbose logging:
uvicorn app:app --reload --log-level debug
```

Step 11: Next Steps

1. **Add Sample PDFs**: Place some test PDF files in the uploads folder
2. **Customize UI**: Modify templates and static files for your preferred design
3. **Add Features**: Implement voice queries, multi-language support, etc.
4. **Optimize Performance**: Adjust chunk sizes and embedding models
5. **Add Authentication**: Implement user management if needed

Additional Resources

- FastAPI Documentation
- ChromaDB Documentation
- LangChain Documentation
- OpenAI API Documentation

Support

If you encounter any issues:

1. Check the console output for error messages
2. Verify all dependencies are installed correctly

3. Ensure API keys are properly configured
4. Check that all required directories exist
5. Review the logs for detailed error information