Complete Setup Guide: LangGraph, Qdrant, and FastAPI for PDF Semantic Search

Overview

This guide walks you through setting up a complete system that combines LangGraph, Qdrant vector database, and FastAPI to build an application that can upload PDF documents and perform semantic search on them.

What You'll Build

A complete API service with:

1. **Vector Database**: Qdrant for storing document embeddings
2. **PDF Upload System**: Extract text from PDFs, chunk it, and store embeddings
3. **Semantic Search**: Query documents using natural language
4. **LangGraph Integration**: Build stateful workflows for document processing

Prerequisites

Before you begin, ensure you have:

- Python 3.8 or higher installed
- pip package manager
- Docker installed (for Qdrant)
- Basic knowledge of Python and virtual environments

Step 1: Environment Setup

Create Virtual Environment

```
 python -m venv langgraph-env
source langgraph-env/bin/activate  # On Windows: langgraph-env\Scripts\activate
```

Create Project Structure

```
project/
├── main.py                 # Main FastAPI application file
├── services/               # Business logic services
│   ├── __init__.py
│   ├── pdf_processor.py   # PDF text extraction and chunking
│   ├── embeddings.py      # Embedding generation logic
│   └── vector_store.py    # Qdrant database operations
├── models/                 # Data models
│   └── schemas.py         # Request/response schemas
├── uploads/                # Temporary storage for uploaded PDFs
├── qdrant_storage/         # Qdrant data persistence
├── .env                    # Environment variables
└── requirements.txt        # Dependencies list
```

Step 2: Install All Dependencies

Create Requirements File

Create `requirements.txt` with all necessary packages:

```
# LangGraph and LangChain
langgraph>=0.0.20
langchain>=0.1.0
langchain-openai>=0.0.5
langchain-text-splitters>=0.0.1
langchain-community>=0.0.20

# Qdrant Vector Database
qdrant-client>=1.7.0

# FastAPI and Web Server
fastapi>=0.104.0
uvicorn>=0.24.0
python-multipart>=0.0.6

# PDF Processing
pypdf>=3.17.0

# Embeddings
sentence-transformers>=2.2.2

# Utilities
python-dotenv>=1.0.0
```

Install All Packages

```
pip install -r requirements.txt
```

What each component does:

- **LangGraph**: Build stateful, multi-actor applications with LLMs
- **LangChain**: Framework for developing LLM applications
- **Qdrant**: Vector similarity search engine
- **FastAPI**: Modern web framework for building APIs
- **Uvicorn**: ASGI server to run FastAPI
- **pypdf**: Extract text from PDF files
- **sentence-transformers**: Generate text embeddings
- **python-dotenv**: Manage environment variables

Step 3: Setup Qdrant Database

Option A: Using Docker (Recommended)

1. **Pull Qdrant Docker image:**

```
docker pull qdrant/qdrant
```

2. **Run Qdrant container:**

```
docker run -p 6333:6333 -p 6334:6334 \
    -v $(pwd)/qdrant_storage:/qdrant/storage:z \
    qdrant/qdrant
```

3. **Verify Qdrant is running:**
    - Open browser: `http://localhost:6333/dashboard`
    - You should see the Qdrant dashboard

Option B: Using Qdrant Cloud

1. Sign up at cloud.qdrant.io
2. Create a new cluster
3. Note your cluster URL and API key

Option C: In-Memory (Development Only)

Use Qdrant client in memory mode:

- No installation needed
- Data is lost when application stops
- Good for testing only

Step 4: Configure Environment Variables

Create a `.env` file in your project root:

```
# OpenAI API (if using OpenAI embeddings)
OPENAI_API_KEY=your_openai_api_key_here

# Qdrant Configuration
QDRANT_URL=http://localhost:6333
QDRANT_API_KEY=your_qdrant_api_key  # Only for Qdrant Cloud

# Embedding Model
EMBEDDING_MODEL=all-MiniLM-L6-v2

# Collection Settings
COLLECTION_NAME=pdf_documents

# Chunking Parameters
CHUNK_SIZE=1000
CHUNK_OVERLAP=200
```

Step 5: Implement Core Services

You need to create four main service modules:

Embedding Service (`services/embeddings.py`)

Purpose: Generate vector embeddings from text

What to implement:

- Initialize sentence-transformer model (e.g., all-MiniLM-L6-v2)
- Method to generate embedding for single text
- Method to generate embeddings for batch of texts
- Return vector dimensions for collection setup

Key concepts:

- **all-MiniLM-L6-v2**: Fast, 384 dimensions, good for general use
- **all-mpnet-base-v2**: Higher quality, 768 dimensions, slower

PDF Processor Service (`services/pdf_processor.py`)

Purpose: Extract and chunk text from PDFs

What to implement:

- Extract text from PDF files using pypdf
- Split text into manageable chunks (recommended: 1000 chars)
- Add overlap between chunks (recommended: 200 chars) for context
- Attach metadata (filename, chunk ID, total chunks, source)
- Return list of document chunks ready for embedding

Key concepts:

- **Chunking**: Breaking large documents into smaller pieces
- **Overlap**: Maintaining context between consecutive chunks
- **Metadata**: Additional information about each chunk

Vector Store Service (`services/vector_store.py`)

Purpose: Manage Qdrant database operations

What to implement:

- Connect to Qdrant (local or cloud)
- Create collection with correct vector dimensions
- Insert documents with embeddings and metadata
- Perform similarity search with configurable parameters
- Return search results with scores and metadata

Key concepts:

- **Collection**: Container for vectors in Qdrant
- **Cosine Similarity**: Measure of vector similarity (0 to 1)
- **Score Threshold**: Minimum similarity score for results

Data Schemas (`models/schemas.py`)

Purpose: Define API request and response structures

What to implement:

- SearchQuery model (query text, top_k, score_threshold)
- SearchResult model (text, score, metadata)
- UploadResponse model (message, filename, chunks_created)

Step 6: Build FastAPI Application

In `main.py`, implement the following:

Initialize Application

- Create FastAPI app instance with title and description
- Add CORS middleware for cross-origin requests
- Load environment variables from .env file
- Initialize all services (embeddings, PDF processor, vector store)

Startup Event

On application startup:

- Connect to Qdrant database
- Create collection if it doesn't exist
- Set vector dimensions based on embedding model

API Endpoints

Root Endpoint (`GET /`)

- Return API information
- List available endpoints

Health Check (`GET /health`)

- Verify API is running
- Check Qdrant connection status
- List available collections
- Return status response

Upload PDF (`POST /upload`)

Workflow:

1. Accept PDF file upload (multipart/form-data)
2. Validate file is PDF format
3. Save file temporarily
4. Extract text using PDF processor
5. Chunk text with metadata
6. Generate embeddings for all chunks
7. Store vectors in Qdrant with metadata
8. Delete temporary file
9. Return response with chunk count and status

Parameters:

- File: PDF document (required)

Response:

- Success message
- Filename
- Number of chunks created
- Collection name

Semantic Search (`POST /search`)

Workflow:

1. Accept search query as JSON
2. Generate embedding for query
3. Search Qdrant for similar vectors
4. Apply score threshold filter

5. Return top-k results with metadata

Parameters:

- query: Search text (required)
- top_k: Number of results (optional, default: 5)
- score_threshold: Minimum similarity (optional, default: 0.7)

Response:

- List of matching chunks
- Similarity scores
- Document metadata (filename, chunk ID)

Step 7: Run the Application

Start Qdrant (if using Docker)

```
docker run -p 6333:6333 -p 6334:6334 \
    -v $(pwd)/qdrant_storage:/qdrant/storage:z \
    qdrant/qdrant
```

Start FastAPI Server

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

Command options:

- `main:app` : Points to app object in main.py
- `--reload` : Auto-reload on code changes (dev only)
- `--host 0.0.0.0` : Accept external connections
- `--port 8000` : Run on port 8000

Access the API

- **API Base**: `http://localhost:8000`
- **Interactive Docs**: `http://localhost:8000/docs` (Swagger UI)
- **Alternative Docs**: `http://localhost:8000/redoc`
- **Qdrant Dashboard**: `http://localhost:6333/dashboard`

Step 8: Test Your API

Using Swagger UI (Easiest)

1. Go to `http://localhost:8000/docs`
2. Click on **POST /upload**
    - Click "Try it out"
    - Choose a PDF file
    - Click "Execute"
    - Verify success response
3. Click on **POST /search**
    - Click "Try it out"
    - Enter search query
    - Adjust top_k and score_threshold
    - Click "Execute"
    - View results with scores

Using cURL (Command Line)

Health check:

```
curl http://localhost:8000/health
```

Upload PDF:

```
curl -X POST "http://localhost:8000/upload" \
  -H "accept: application/json" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@/path/to/document.pdf"
```

Search:

```
curl -X POST "http://localhost:8000/search" \
  -H "Content-Type: application/json" \
  -d '{
    "query": "What is machine learning?",
    "top_k": 5,
    "score_threshold": 0.7
  }'
```

Using Python

```
pip install requests
```

Then create a test script to upload PDFs and perform searches.

How It All Works Together

Document Upload Flow

1. User uploads PDF via `/upload` endpoint
2. FastAPI receives file and saves temporarily
3. PDF Processor extracts text from PDF
4. Text Splitter chunks text into pieces with overlap
5. Embedding Service generates vectors for each chunk
6. Vector Store saves embeddings to Qdrant with metadata
7. Temporary file deleted, response sent to user

Search Flow

1. User sends query via `/search` endpoint
2. Embedding Service converts query to vector
3. Vector Store searches Qdrant using cosine similarity
4. Qdrant returns most similar chunks with scores
5. Results filtered by score threshold
6. Top-k results returned to user with metadata

Key Concepts Explained

Chunking Strategy

- **Why**: Large documents don't fit in embedding models
- **Size**: 1000 characters balances context and specificity
- **Overlap**: 200 characters maintains continuity between chunks

Vector Embeddings

- **What**: Numerical representations of text meaning
- **Why**: Enable semantic similarity comparison
- **Dimensions**: Model-specific (384 or 768 common)

Semantic Search

- **How**: Compares query vector to document vectors
- **Metric**: Cosine similarity (0 = different, 1 = identical)
- **Ranking**: Higher scores = more relevant results

Qdrant Collections

- **Purpose**: Organize vectors by topic/type
- **Schema**: Fixed vector dimensions, flexible metadata
- **Indexing**: Enables fast similarity search

LangGraph Integration (Optional)

To add LangGraph workflows for document processing:

What LangGraph Adds

- **State Management**: Track document processing status
- **Multi-step Workflows**: Chain processing steps

- **Error Handling**: Retry failed operations
- **Conditional Logic**: Route based on document type

Example Use Cases

- Process documents through multiple stages
- Implement approval workflows
- Add human-in-the-loop validation
- Create complex document analysis pipelines

Troubleshooting

Qdrant Connection Issues

- **Check Docker**: `docker ps` to verify container running
- **Check Port**: Ensure 6333 isn't used by another service
- **Check Firewall**: Allow connections to port 6333

Upload Failures

- **Verify python-multipart**: Must be installed for file uploads
- **Check Permissions**: uploads/ directory must be writable
- **File Size**: Large PDFs may timeout (increase timeout settings)

No Search Results

- **Lower Threshold**: Try score_threshold = 0.5 or lower
- **Verify Upload**: Check collection has documents
- **Try Exact Match**: Search for text you know is in documents

Slow Performance

- **Use Faster Model**: Switch to all-MiniLM-L6-v2
- **Reduce Chunks**: Increase chunk_size to create fewer vectors
- **Add Caching**: Cache frequent queries
- **Scale Qdrant**: Use Qdrant Cloud for production

Import Errors

- **Activate Environment**: Ensure virtual environment is active
- **Reinstall**: `pip install --upgrade -r requirements.txt`
- **Check Python Version**: Must be 3.8 or higher

Next Steps and Enhancements

Security

- Add API key authentication
- Implement user authentication (JWT tokens)
- Add rate limiting to prevent abuse
- Validate and sanitize file uploads

Features

- **Batch Upload**: Process multiple PDFs at once
- **Delete Endpoint**: Remove documents from database
- **Metadata Filtering**: Search within specific documents
- **Pagination**: Return results in pages
- **Format Support**: Add DOCX, TXT, HTML support
- **Summarization**: Generate summaries of search results

Production Readiness

- **Docker Compose**: Containerize entire stack
- **Logging**: Add comprehensive logging
- **Monitoring**: Track API performance and errors
- **Testing**: Add unit and integration tests
- **CI/CD**: Automate deployment pipeline
- **Backup**: Regular Qdrant database backups

Advanced Features

- **Hybrid Search**: Combine keyword and semantic search
- **Reranking**: Improve result relevance with reranker models

- **Multi-language**: Support documents in various languages
- **OCR Integration**: Extract text from scanned PDFs
- **Real-time Updates**: Watch folders for new documents

Resources and Documentation

- **LangGraph**: https://langchain-ai.github.io/langgraph/
- **Qdrant**: https://qdrant.tech/documentation/
- **FastAPI**: https://fastapi.tiangolo.com/
- **Sentence Transformers**: https://www.sbert.net/
- **LangChain**: https://python.langchain.com/