Untitled 37

RAG Service API Guide

This guide explains how to use the RAG service API endpoints for document ingestion and querying.

Base URL

```
http://localhost:8000
```

For production, replace with your deployed service URL.

📥 Ingestion Endpoint

POST `/ingest`

Ingest documents (PDF, DOCX, TXT, or video files) into the RAG system.

Request Format

This endpoint accepts **multipart/form-data** with the following fields:

Field	Type	Required	Description
`file`	File	✅ Yes	The file to ingest (PDF, DOCX, TXT, MP4)
`course_id`	String	✅ Yes	Course identifier for multi-tenancy
`module_id`	String	✅ Yes	Module identifier within the course
`source_type`	String	✅ Yes	Source type: `pdf`, `docx`, `txt`, `video`, or `notes`
`video_id`	String	❌ No	Video ID (required if source_type is `video`)
`notes_id`	String	❌ No	Notes ID (required if source_type is `notes`)

Example: cURL

```
# Ingest a PDF file
```

```
curl -X POST "http://localhost:8000/ingest" \

  -F "course_id=DL101" \

  -F "module_id=M03" \

  -F "source_type=pdf" \

  -F "file=@lecture_notes.pdf"



# Ingest a video file

curl -X POST "http://localhost:8000/ingest" \

  -F "course_id=DL101" \

  -F "module_id=M02" \

  -F "source_type=video" \

  -F "video_id=video_123" \

  -F "file=@lecture_video.mp4"
```

Example: JavaScript (Fetch API)

```
const formData = new FormData();

formData.append('file', fileInput.files[0]);

formData.append('course_id', 'DL101');

formData.append('module_id', 'M03');

formData.append('source_type', 'pdf');



const response = await fetch('http://localhost:8000/ingest', {

  method: 'POST',

  body: formData
});



const result = await response.json();

console.log(result);
```

Example: Python (Requests)

```
import requests



files = {'file': open('lecture_notes.pdf', 'rb')}

data = {

    'course_id': 'DL101',

    'module_id': 'M03',

    'source_type': 'pdf'

}



response = requests.post('http://localhost:8000/ingest', files=files,
data=data)

print(response.json())
```

Response

```
{

  "job_id": "550e8400-e29b-41d4-a716-446655440000",

  "status": "completed",

  "message": "Successfully ingested 15 chunks",

  "chunks_count": 15
}
```

Response Fields:

- `job_id` : Unique identifier for the ingestion job
- `status` : Job status (`queued` , `processing` , `completed` , or `failed`)
- `message` : Human-readable status message
- `chunks_count` : Number of text chunks created (only present for completed jobs)

🔍 Query Endpoint

POST `/query`

Query the RAG system with optional filtering by course and module.

Request Format

This endpoint accepts **JSON** payload:

Field	Type	Required	Default	Description
`query`	String	✅ Yes	-	The question to ask
`course_id`	String	❌ No	null	Filter results by specific course
`module_id`	String	❌ No	null	Filter results by specific module
`top_k`	Integer	❌ No	5	Number of chunks to retrieve (1-100)
`full_context`	Boolean	❌ No	false	If true, retrieve ALL chunks for the module
`include_sources`	Boolean	❌ No	true	If false, don't return source chunks

Example: cURL

```
# Basic query

curl -X POST "http://localhost:8000/query" \

  -H "Content-Type: application/json" \

  -d '{
    "query": "Explain the attention mechanism in transformers"
  }'

# Query with course and module filtering
curl -X POST "http://localhost:8000/query" \
  -H "Content-Type: application/json" \
  -d '{
    "query": "Explain the attention mechanism",
    "course_id": "DL101",
    "module_id": "M03",
    "top_k": 5
  }'
```

Example: JavaScript (Fetch API)

```
const queryData = {
  query: "Explain the attention mechanism in transformers",
  course_id: "DL101",
  module_id: "M03",
  top_k: 5
};

const response = await fetch('http://localhost:8000/query', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'

  },

  body: JSON.stringify(queryData)

});



const result = await response.json();

console.log(result);
```

Example: Python (Requests)

```
import requests



payload = {

    "query": "Explain the attention mechanism in transformers",

    "course_id": "DL101",

    "module_id": "M03",

    "top_k": 5

}



response = requests.post('http://localhost:8000/query', json=payload)

print(response.json())
```

Response

```
{

  "answer": "The attention mechanism allows the model to focus on different parts of the input sequence when generating each output token. It computes
attention scores...",
  "sources": [
    {
      "chunk_id": "550e8400-e29b-41d4-a716-446655440000",
      "score": 0.89,
      "source_uri": "course/DL101/module/M03/lecture_notes.pdf",
      "source_type": "pdf",
      "text_preview": "Attention mechanism is a key component of transformer
architecture...",
      "start_time_seconds": null,
      "end_time_seconds": null
    }
  ],
  "debug": {
    "search_latency_ms": 45.2,
    "llm_latency_ms": 1234.5,
    "total_latency_ms": 1279.7,
    "chunks_retrieved": 5,
    "tokens_used": 450,
    "cache_hit": false
  }
}
```

Response Fields:

- `answer` : The AI-generated answer based on retrieved context
- `sources` : List of source chunks used to generate the answer

- `chunk_id` : Unique identifier for the chunk
- `score` : Relevance score (0-1, higher is better)
- `source_uri` : Path/identifier of the source document
- `source_type` : Type of source (`pdf` , `video` , `notes` , etc.)
- `text_preview` : Preview of the chunk text (max 200 chars)
- `start_time_seconds` / `end_time_seconds` : Time markers for video sources

* `debug` : Performance and debugging information

- `search_latency_ms` : Time to retrieve relevant chunks
- `llm_latency_ms` : Time for LLM to generate answer
- `total_latency_ms` : Total query processing time
- `chunks_retrieved` : Number of chunks retrieved
- `tokens_used` : Estimated tokens used by LLM
- `cache_hit` : Whether the result was served from cache

🏥 Health Check Endpoint

GET `/health`

Check the health status of the RAG service and its dependencies.

Example: cURL

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

Response

```
{
  "status": "healthy",
  "version": "1.0.0",
  "qdrant_connected": true,

  "redis_connected": true

}
```

🧪 Interactive API Documentation

FastAPI provides interactive API documentation at:

- **Swagger UI**: http://localhost:8000/docs
- **ReDoc**: http://localhost:8000/redoc

These interfaces allow you to test the API endpoints directly from your browser.

🔧 Common Use Cases

1. Ingest Course Materials

```
# Ingest lecture PDF

curl -X POST "http://localhost:8000/ingest" \

  -F "course_id=CS101" \

  -F "module_id=intro" \

  -F "source_type=pdf" \

  -F "file=@intro_lecture.pdf"



# Ingest lecture video

curl -X POST "http://localhost:8000/ingest" \

  -F "course_id=CS101" \

  -F "module_id=intro" \

  -F "source_type=video" \
  -F "video_id=vid_001" \
  -F "file=@intro_video.mp4"
```

2. Query Entire Course

```
curl -X POST "http://localhost:8000/query" \
  -H "Content-Type: application/json" \
  -d '{
    "query": "What are the main topics covered?",
    "course_id": "CS101"
  }'
```

3. Query Specific Module

```
curl -X POST "http://localhost:8000/query" \
  -H "Content-Type: application/json" \
  -d '{
    "query": "Explain the key concepts",
    "course_id": "CS101",
    "module_id": "intro",
    "top_k": 10
  }'
```

4. Get Full Module Context

```
curl -X POST "http://localhost:8000/query" \
  -H "Content-Type: application/json" \

  -d '{

    "query": "Summarize this module",

    "course_id": "CS101",

    "module_id": "intro",

    "full_context": true,

    "include_sources": false

  }'
```

⚠️ Error Handling

The API returns standard HTTP status codes:

- `200` : Success
- `400` : Bad Request (invalid parameters)
- `422` : Validation Error (missing required fields)
- `500` : Internal Server Error

Error response example:

```
{

  "detail": "Invalid source_type. Must be one of: ['pdf', 'docx', 'txt', 'video', 'notes']"

}
```

� Python Helper Functions

Here are reusable Python functions for common operations:

Ingest Notes (PDF, TXT, DOCX)

```python
import requests

import os

import json


def ingest_notes(

    file_path: str,

    course_id: str,

    module_id: str,

    notes_id: str,

    base_url: str = "http://localhost:8000"

) -> dict:

    """

    Ingest a notes file (PDF, TXT, or DOCX) into the RAG system.

    Automatically detects file type from extension.

    """

    if not os.path.exists(file_path):

        raise FileNotFoundError(f"File not found: {file_path}")

    # Auto-detect source type from file extension

    _, ext = os.path.splitext(file_path)

    ext = ext.lower()

    ext_to_source_type = {".pdf": "pdf", ".txt": "txt", ".docx": "docx"}

    source_type = ext_to_source_type.get(ext)

    if not source_type:

        raise ValueError(f"Unsupported file type: {ext}")
    with open(file_path, "rb") as f:

        files = {"file": (os.path.basename(file_path), f)}

        data = {

            "course_id": course_id,

            "module_id": module_id,

            "source_type": source_type,

            "notes_id": notes_id,

        }

        print(f"Ingesting {os.path.basename(file_path)}...")

        print(f"  Course ID: {course_id}")

        print(f"  Module ID: {module_id}")

        print(f"  Source Type: {source_type}")

        response = requests.post(f"{base_url}/ingest", files=files,
data=data)

    result = response.json()

    print(json.dumps(result, indent=2))

    return result



# Usage

ingest_notes(

    file_path="lecture_notes.pdf",

    course_id="15",

    module_id="9",

    notes_id="4"

)
```

Ingest Video

```python
import requests

import os

import json


def ingest_video(

    file_path: str,

    course_id: str,

    module_id: str,

    video_id: str,

    base_url: str = "http://localhost:8000"

) -> dict:

    """

    Ingest a video file into the RAG system.

    Extracts audio and transcribes automatically.

    """

    if not os.path.exists(file_path):

        raise FileNotFoundError(f"File not found: {file_path}")

    with open(file_path, "rb") as f:

        files = {"file": (os.path.basename(file_path), f)}

        data = {

            "course_id": course_id,

            "module_id": module_id,

            "source_type": "video",
            "video_id": video_id,

        }

        print(f"Ingesting {os.path.basename(file_path)}...")

        print(f"  Course ID: {course_id}")

        print(f"  Module ID: {module_id}")

        print(f"  Video ID: {video_id}")

        print("  (This may take a while for audio transcription...)")

        response = requests.post(f"{base_url}/ingest", files=files,
data=data)

    result = response.json()

    print(json.dumps(result, indent=2))

    return result



# Usage

ingest_video(

    file_path="lecture_video.mp4",

    course_id="15",

    module_id="7",

    video_id="6"

)
```

Query with Filters

```
import requests

import json
```

```python
def query_rag(
    query: str,
    course_id: str = None,
    module_id: str = None,
    top_k: int = 10,
    full_context: bool = False,
    include_sources: bool = True,
    base_url: str = "http://localhost:8000"
) -> dict:
    """
    Query the RAG system with optional filtering.
    """
    payload = {
        "query": query,
        "top_k": top_k,
        "full_context": full_context,
        "include_sources": include_sources
    }
    if course_id:
        payload["course_id"] = course_id
    if module_id:
        payload["module_id"] = module_id
    response = requests.post(f"{base_url}/query", json=payload)
    result = response.json()
    print(json.dumps(result, indent=2))
    return result



# Query specific course and module

query_rag(

    query="What are the percentage of Sinhalese and Indian Tamils in Sri Lanka?",

    course_id="15",

    module_id="7",

    top_k=10

)



# Query without module filter (entire course)

query_rag(

    query="Generate some questions on Sri Lanka?",

    course_id="15",

    top_k=10

)



# Get full context (all chunks) without source details in response

query_rag(

    query="Summarize the entire module",

    course_id="15",

    module_id="7",

    full_context=True,

    include_sources=False
)
```

� 📝 Notes

- **Multi-tenancy**: Use `course_id` and `module_id` to organize content hierarchically
- **File Formats**: Supported formats include PDF, DOCX, TXT, and MP4 (video)
- **Video Processing**: For videos, transcripts are extracted and chunked automatically (may take several minutes)
- **Caching**: Query results are cached in Redis for improved performance
- **Filters**: Combine `course_id` and `module_id` filters for precise content retrieval
- **Full Context Mode**: Use `full_context=True` to retrieve all chunks from a module instead of top-k semantic search
- **Source Toggle**: Set `include_sources=False` when you only need the answer and not the source chunks (useful with full_context)