Hello, Im **Gokulnath,** please find my task submission below.

—

🚀 FastAPI Server for Retrieval Augmented Generation (RAG) System

Task Overview

The goal was to implement a **lightweight FastAPI server** designed for a **Retrieval Augmented Generation (RAG)** system with the following requirements:

- 💾 **ChromaDB** for document ingestion and querying
- 📂 Support for multiple document formats: **PDF, DOC, DOCX, TXT**
- 🧠 Use **sentence-transformers/all-MiniLM-L6-v2** from Hugging Face for embeddings
- ⚡ Ensure **non-blocking API endpoints** and efficient concurrency with FastAPI's async features

---

Task submission - Github repo : https://github.com/Coding-Devil/FastAPI-Server-for-RAG/tree/main

🛠️ My Approach and Implementation

1. FastAPI Server

- Developed a FastAPI server utilizing **asynchronous capabilities** for handling non-blocking operations.
- The server exposes endpoints for:
  - 📥 **Document upload**
  - 🔍 **Querying the database**
  - 📋 **Listing available collections**

2. Document Processing

- Implemented support for various document types including **PDF, DOC, DOCX, and TXT**.
- Asynchronous file handling via **asyncio** ensures smooth, non-blocking processing.

3. Embeddings

- Chose **sentence-transformers/all-MiniLM-L6-v2** from Hugging Face for generating embeddings. This model is lightweight and performs efficiently even on **CPUs**.

4. Vector Database Integration

- Integrated **ChromaDB** as the vector database, allowing persistent storage and querying of document embeddings.

5. RAG System Implementation

- The **RAG server** retrieves relevant document chunks based on user queries, leveraging **sentence-transformers** to generate **contextual answers**.

6. Concurrency & Non-blocking Operations

- For time-consuming tasks like file uploads and document ingestion, used FastAPI's **async features** and Python's **asyncio**.
- Background tasks handle long-running processes like document parsing and embedding generation to keep the API **responsive**.

🔧 Technologies Used

- ⚙️ **FastAPI**: Lightweight server framework
- 🗄️ **ChromaDB**: Persistent vector storage
- 🧑‍🏫 **Sentence-Transformers**: Embedding generation
- 🤖 **Hugging Face Inference API**: For RAG-based text generation
- 📑 **PyPDF2 & python-docx**: Document parsing tools
- 🔄 **Asyncio**: For managing asynchronous operations

📁 Project Structure

```
rag_server/
├── app.py            # Main FastAPI application
├── vector_db.py       # ChromaDB integration logic
├── load_data.py       # Document loading and splitting logic
├── prompts.py         # RAG prompt generation
├── utils.py           # Utility functions (e.g., async helpers)
├── ingest.py          # Document ingestion logic
├── COLLECTIONS.txt    # List of document collections
└── data/              # Directory for uploaded documents
```

✅ Meeting Expectations

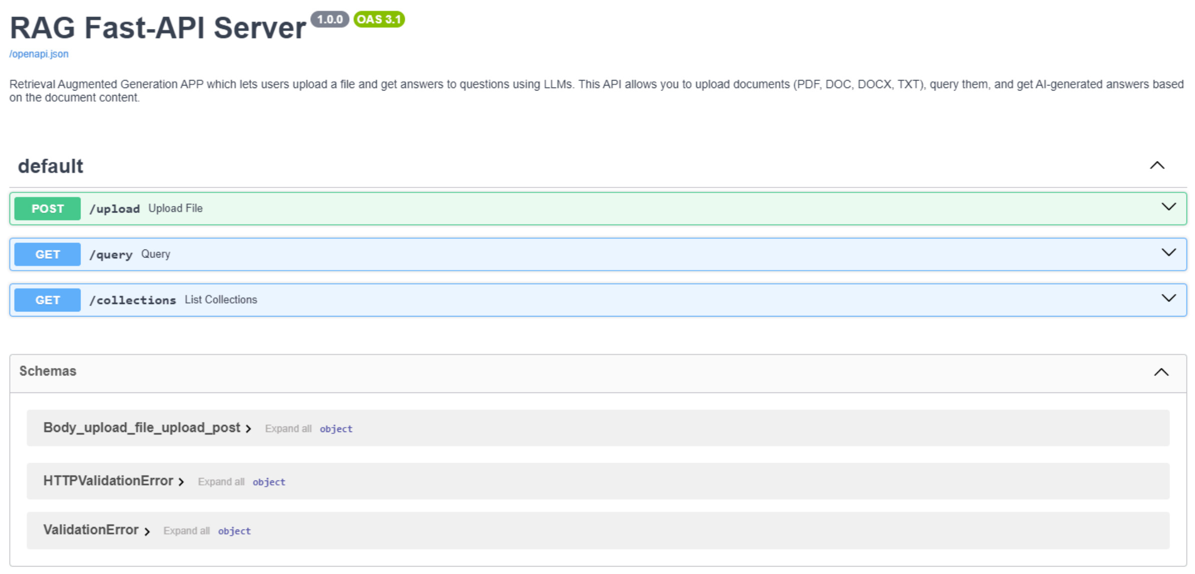
- 💡 **Lightweight Server**: FastAPI ensures efficient, lightweight performance
- 🗃️ **ChromaDB Integration**: Persistent vector storage with querying
- 📂 **Document Support**: Handles **PDF** and **DOCX**, extendable to other formats like **HTML**
- 🧠 **Embeddings**: Using **sentence-transformers** for efficient embeddings
- 🚀 **Non-blocking API**: FastAPI's async capabilities guarantee non-blocking endpoints
- 🔄 **Concurrency**: Efficient handling of concurrent requests through **asyncio**

🎯 Results and Demonstration

The system successfully:

1. Uploads and processes multiple document formats.
2. Efficiently queries stored document collections using embeddings.
3. Provides accurate, context-aware responses via RAG, leveraging pre-trained NLP models.

4. Is scalable and can handle various document collections with ease.



RAG Fast-API Server 1.0.0 OAS 3.1

/openapi.json

Retrieval Augmented Generation APP which lets users upload a file and get answers to questions using LLMs. This API allows you to upload documents (PDF, DOC, DOCX, TXT), query them, and get AI-generated answers based on the document content.

default

POST /upload Upload File

GET /query Query

GET /collections List Collections

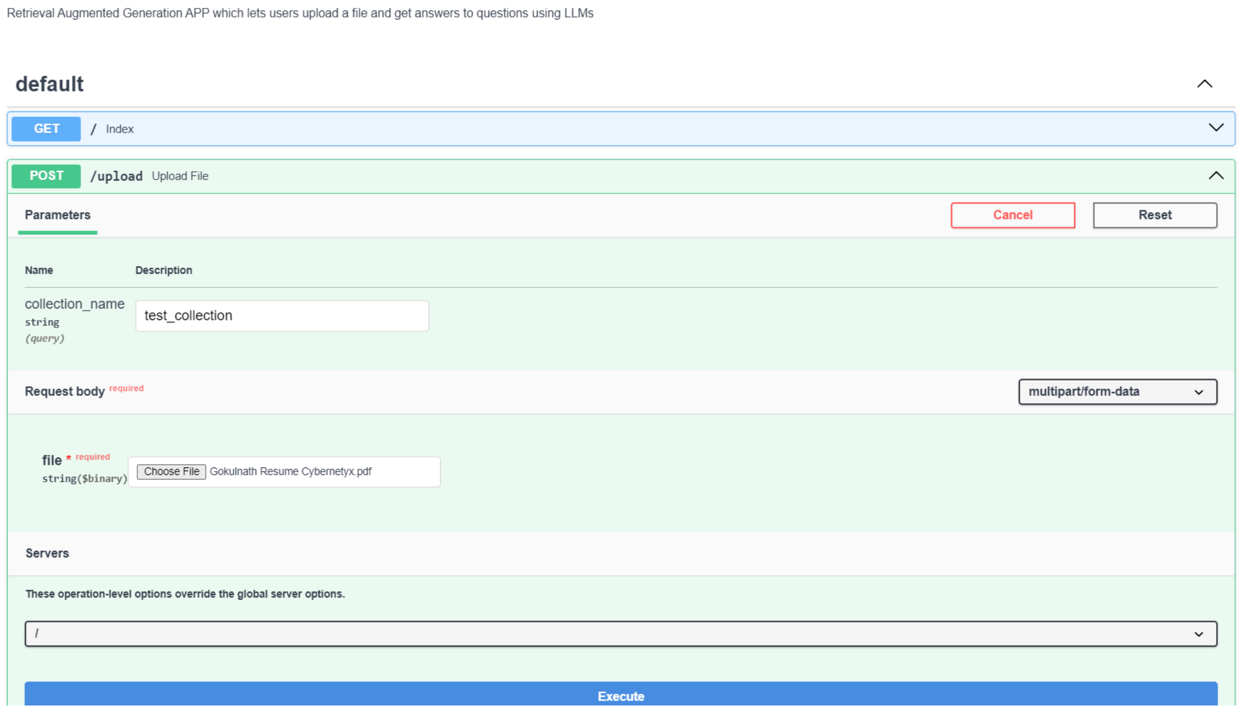
Schemas

Body_upload_file_upload_post > Expand all object

HTTPValidationError > Expand all object

ValidationError > Expand all object

Ingest :



Retrieval Augmented Generation APP which lets users upload a file and get answers to questions using LLMs

default

GET / Index

POST /upload Upload File

Parameters

Name	Description
collection_name string (query)	test_collection

Request body required — multipart/form-data

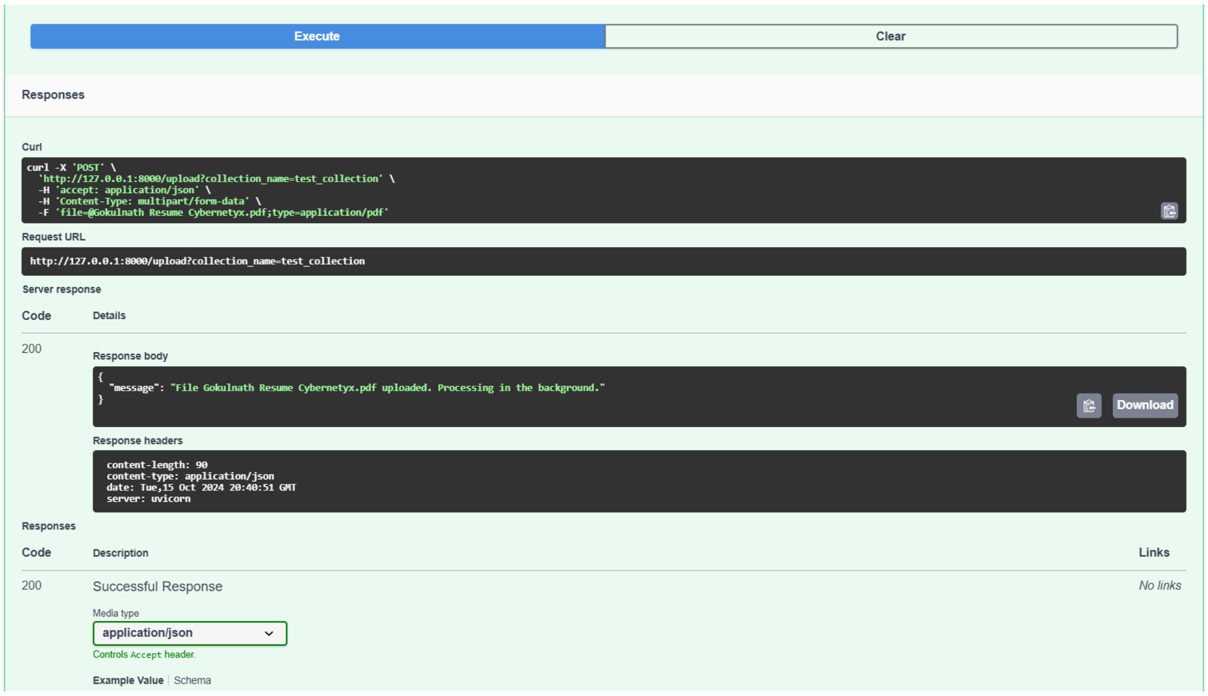
file * required string($binary): Choose File Gokulnath Resume Cybernetyx.pdf

Servers

These operation-level options override the global server options.

/

Upload:



Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/upload?collection_name=test_collection' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@Gokulnath Resume Cybernetyx.pdf;type=application/pdf'
```

Request URL

```
http://127.0.0.1:8000/upload?collection_name=test_collection
```

Server response

Code	Details
200	Response body

```
{
  "message": "File Gokulnath Resume Cybernetyx.pdf uploaded. Processing in the background."
}
```

Response headers

```
 content-length: 90
 content-type: application/json
 date: Tue,15 Oct 2024 20:40:51 GMT
 server: uvicorn
```

Responses

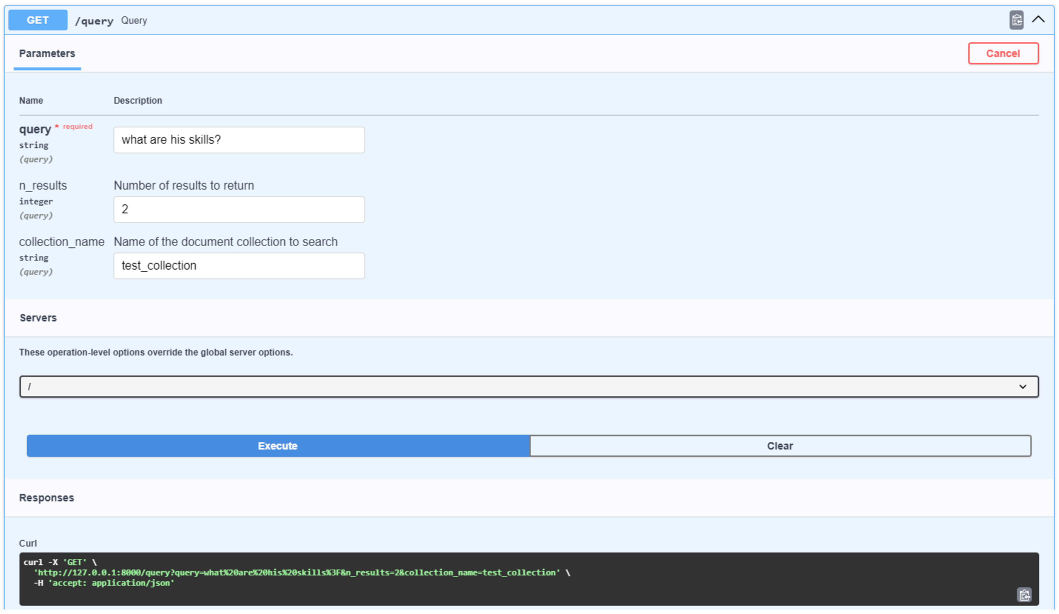
Code	Description	Links
200	Successful Response	*No links*

Media type: application/json

Controls `Accept` header.

Example Value | Schema

Query :



`/query` Query

Parameters

Name	Description
query * required string *(query)*	what are his skills?
n_results integer *(query)*	Number of results to return 2
collection_name string *(query)*	Name of the document collection to search test_collection

Servers

These operation-level options override the global server options.

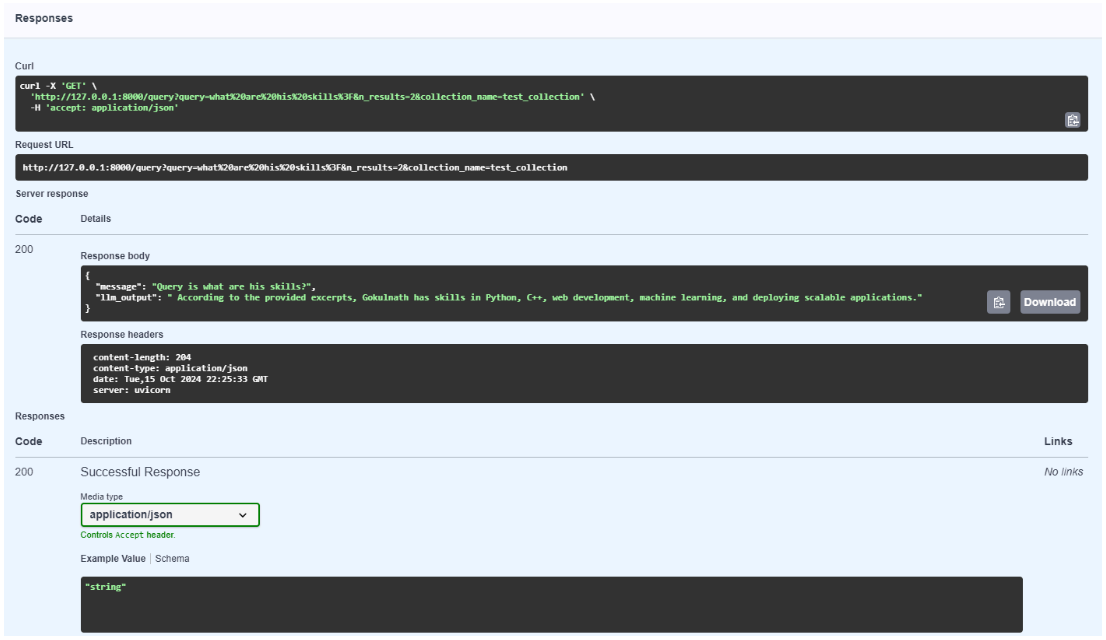
/

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/query?query=what%20are%20his%20skills%3F&n_results=2&collection_name=test_collection' \
  -H 'accept: application/json'
```

Output :



```
curl -X 'GET' \
  'http://127.0.0.1:8000/query?query=what%20are%20his%20skills%3F&n_results=2&collection_name=test_collection' \
  -H 'accept: application/json'
```

```
http://127.0.0.1:8000/query?query=what%20are%20his%20skills%3F&n_results=2&collection_name=test_collection
```

```
{
  "message": "Query is what are his skills?",
  "llm_output": " According to the provided excerpts, Gokulnath has skills in Python, C++, web development, machine learning, and deploying scalable applications."
}
```

```
content-length: 204
content-type: application/json
date: Tue,15 Oct 2024 22:25:33 GMT
server: uvicorn
```

```
"string"
```

Conclusion

In conclusion, this project demonstrates the successful creation of a robust, scalable, and efficient RAG system that adheres to all the outlined requirements. The implementation makes use of modern Python async programming, advanced NLP models from Hugging Face, and effective vector-based information retrieval via ChromaDB.