**Project-2: RAG Based Chatbot And Create FastAPI**

**Submitted By: Rajesh Kumar Sharma**

**Date: 29-Nov-24**

**Project-2:** **Develop a RAG based Chatbot for the given documents and create a FastAPI**

**Chatbot:** Artificial intelligence is used to construct a computer program known as "a chatbot" that simulates human chats with users. It employs a technique known as NLP to comprehend the user's inquiries and offer pertinent information. Chatbots have various functions in customer service, information retrieval, and personal support.

**Limitations of Chatbot:** Chatbot have some limitations to be aware of:

* Lack of semantic understanding: Chatbots may require assistance comprehending the discourse, which could result in misinterpretation or incorrect responses.
* Dependency on training [data:](https://www.simplilearn.com/what-is-data-article) The calibre and volume of training data greatly impact the chatbot performance. There may be a need for more accurate or biased training data, which can result in incorrect responses.
* Handling complicated queries: Chatbots could encounter questions beyond simple pattern matching and call for greater comprehension or deductive reasoning.

We want the installations to do in one shot, we can prepare a text file: requirements.txt with the below contents:

Run the below command for the installation:

***pip install -r requirements.txt***

Requirements.txt

PyPDF2

langchain

huggingface\_hub

langchain-huggingface

langchain\_community

langchain\_core

transformers

torch

faiss-cpu

fastapi

uvicorn

huggingface\_hub

Spacy

python -m spacy download en\_core\_web\_lg

**Code Link:** <https://github.com/rajeshksharmasls/GenAI/blob/master/GenAI/Chatbot%20RAG%20FastAPI/main.py>

**Dataset:** <https://drive.google.com/drive/folders/1VrQ0ZTDjY6E5QhmMteA7fD0FtBOwVMe9>

**Process:** Below is the summarization of the code:

This code defines a **FastAPI-based web application** for creating a **retrieval-augmented generation (RAG) system** that allows users to ingest PDFs, create a searchable vector database, and query it.

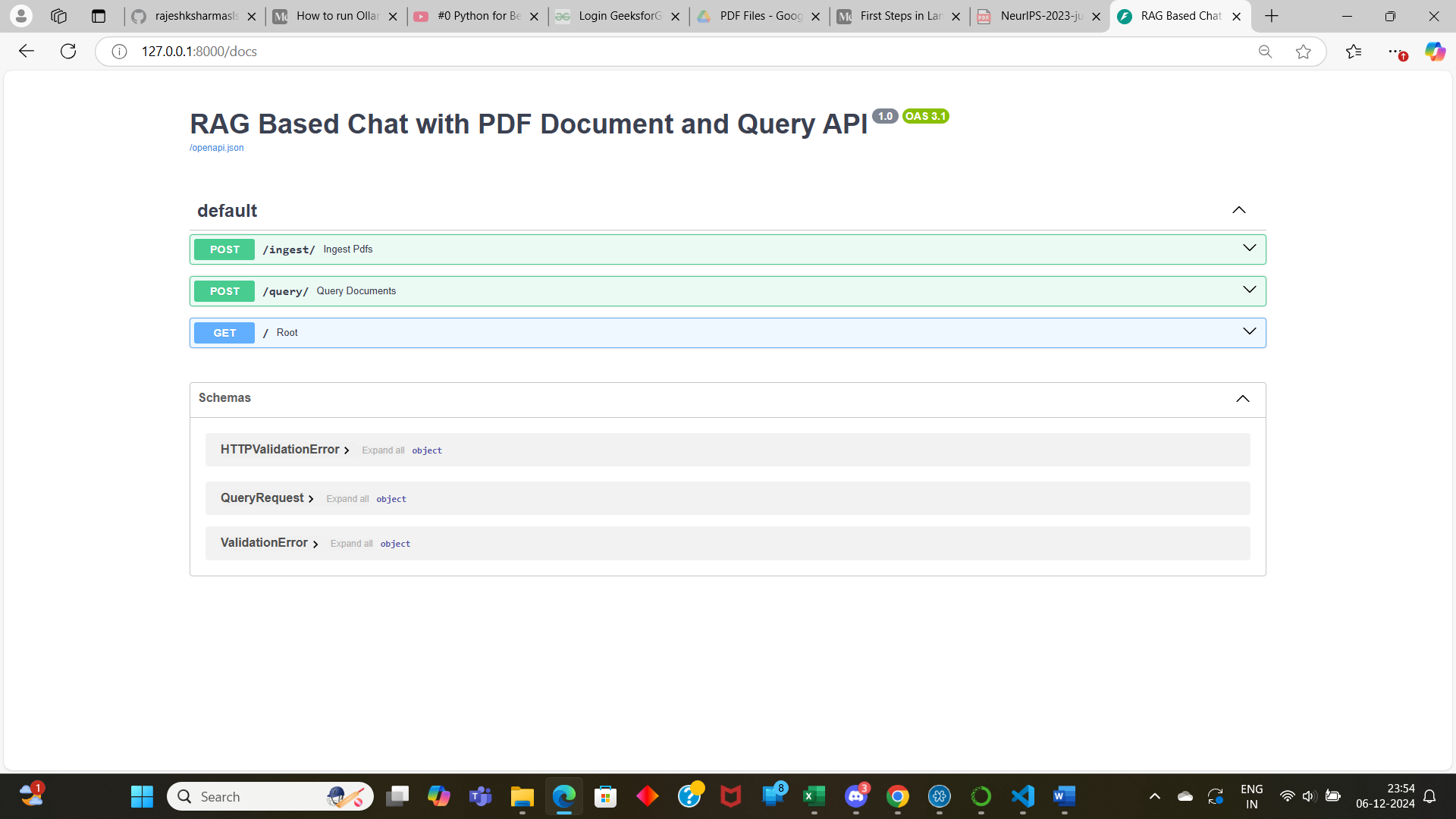
**Key Functionalities:**

1. **PDF Ingestion:**
   * **Endpoint:** /ingest/ (POST)
   * **Description:**
     + Loads PDF files from the PDFFILES directory.
     + Splits their content into smaller chunks using a text splitter.
     + Generates **text embeddings** using SpaCy.
     + Stores these embeddings in a **FAISS vector database**, saved locally for retrieval.
2. **Querying Documents:**
   * **Endpoint:** /query/ (POST)
   * **Description:**
     + Accepts a user query.
     + Loads the FAISS vector database and retrieves relevant document chunks based on embeddings.
     + Returns the text content of the relevant documents.
3. **Root Endpoint:**
   * **Endpoint:** / (GET)
   * **Description:** Simple welcome message for API health check.

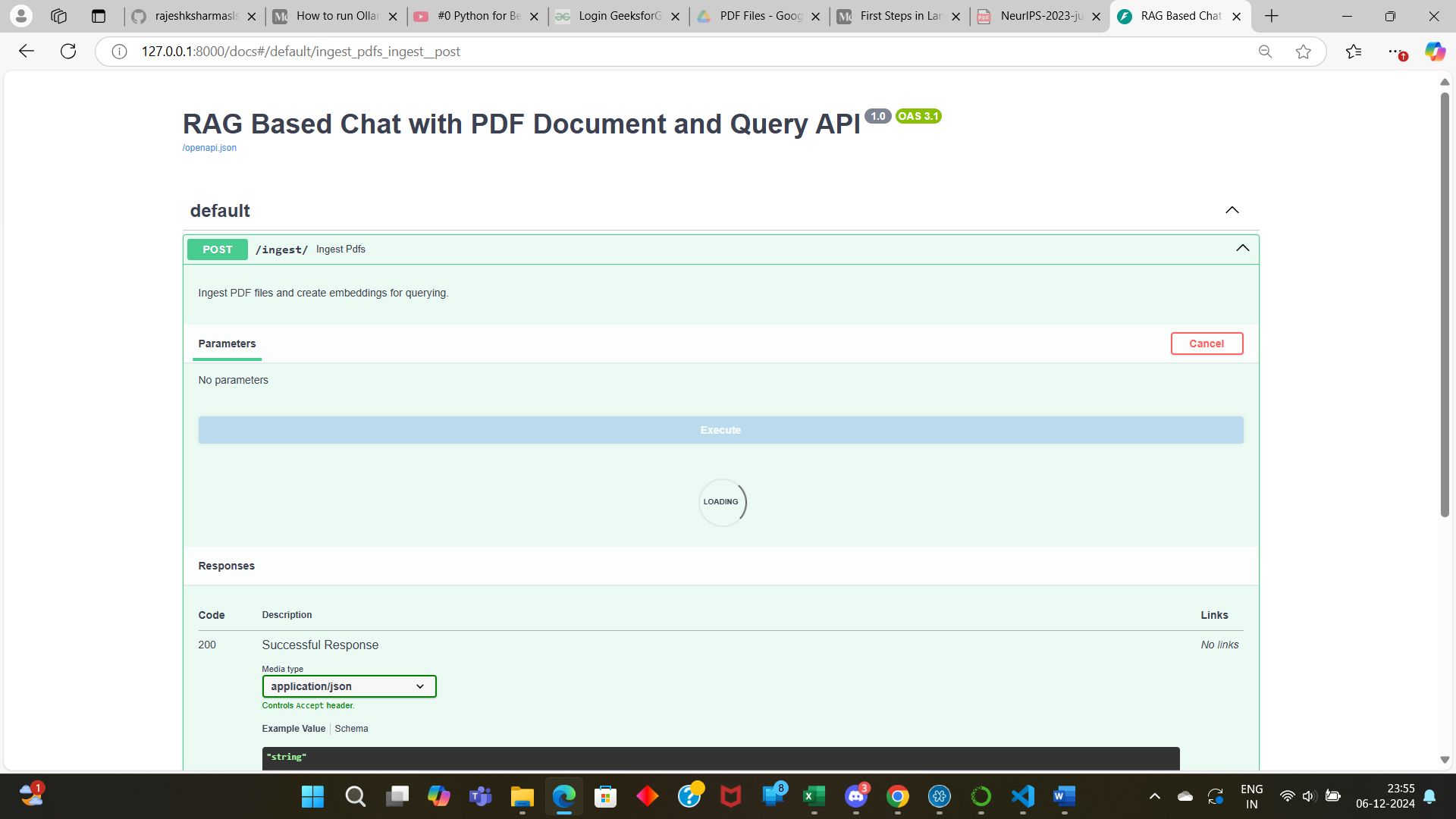
**Technologies and Libraries:**

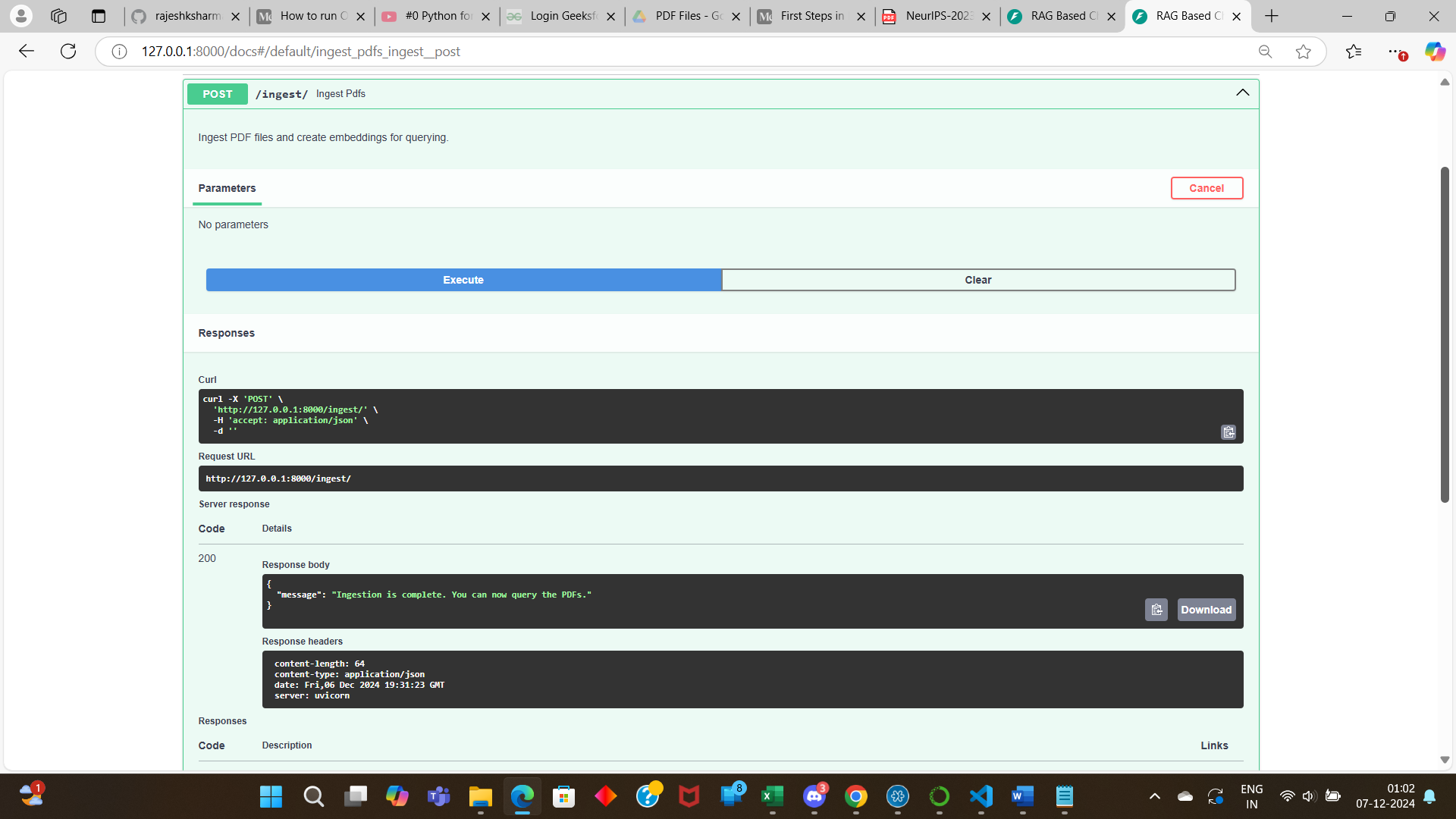
1. **FastAPI:** For creating API endpoints.
2. **PyPDFLoader:** For loading and parsing PDF files.
3. **LangChain:**
   * **Document Processing:** Manages text chunks and metadata.
   * **FAISS Integration:** Enables efficient vector-based storage and retrieval.
4. **SpaCy Embeddings:** For text embedding generation.
5. **Transformers (Hugging Face):**
   * Qwen model (Qwen2.5-0.5B-Instruct): Loaded for potential future use in inference.
6. **Environment Variables:** Managed using dotenv.

**Snapshots of the conversations with the Chatbot:**

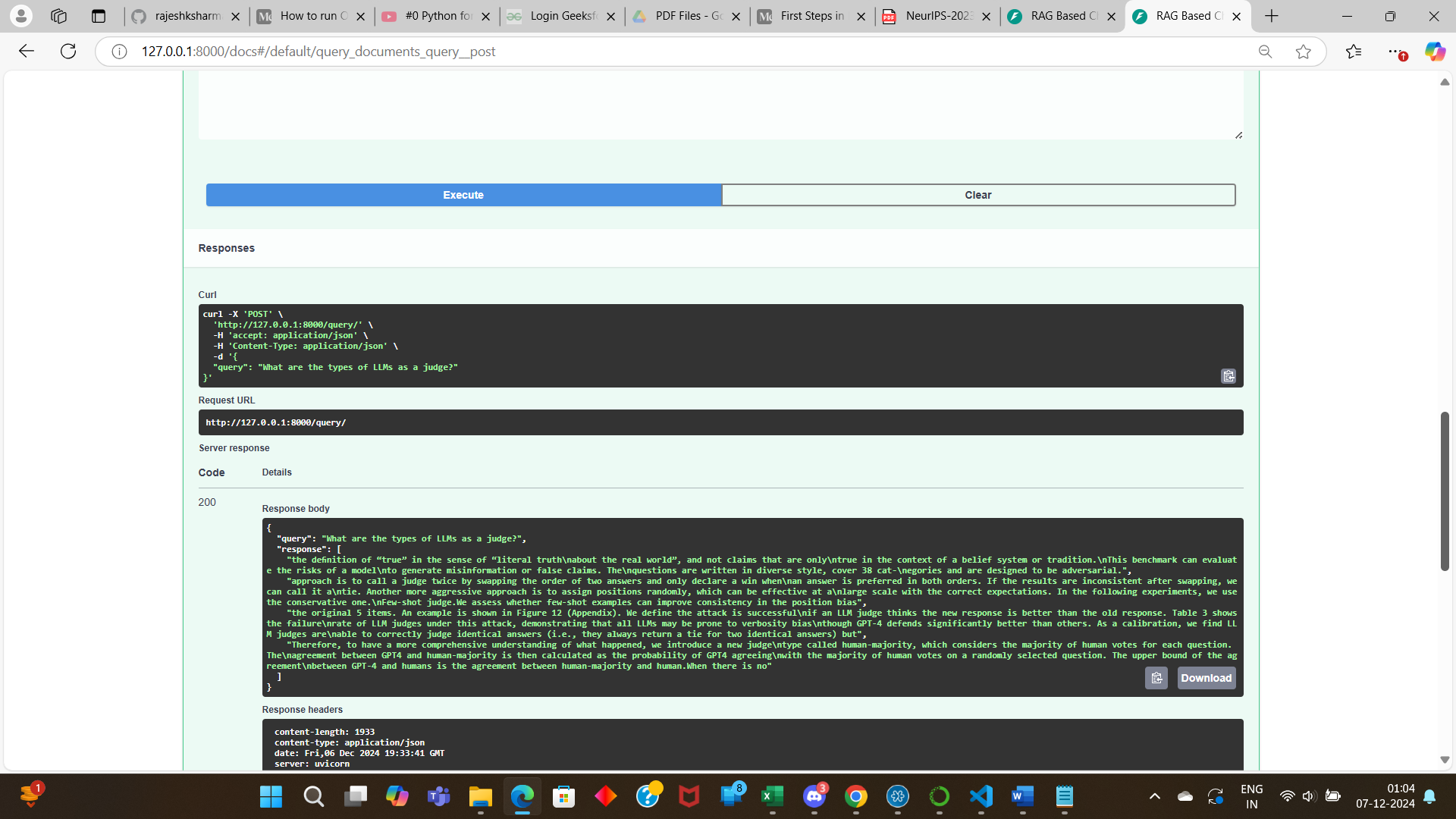
****

**Loading the PDF files:**

****

****

**Response:**

****

**Conclusion:** Hence, we can conclude that even the API version of the RAG chatbot is working fine.