

FASTAPI RAG BACKEND

Introduction

The FastAPI RAG Backend is a system that allows users to chat with their documents. Instead of reading a PDF, the user can simply ask a question, and the AI finds the answer.

How it works:

1. Ingestion: It takes document text and converts it into "Vectors" (numbers).
2. Storage: These vectors are stored in Qdrant (running in Docker).
3. Retrieval: When a user asks a question, the system searches the vectors in Qdrant to find the specific answer from the document.

System Workflow

Document ingestion:

Documents

POST /documents/upload Upload Document

Parameters

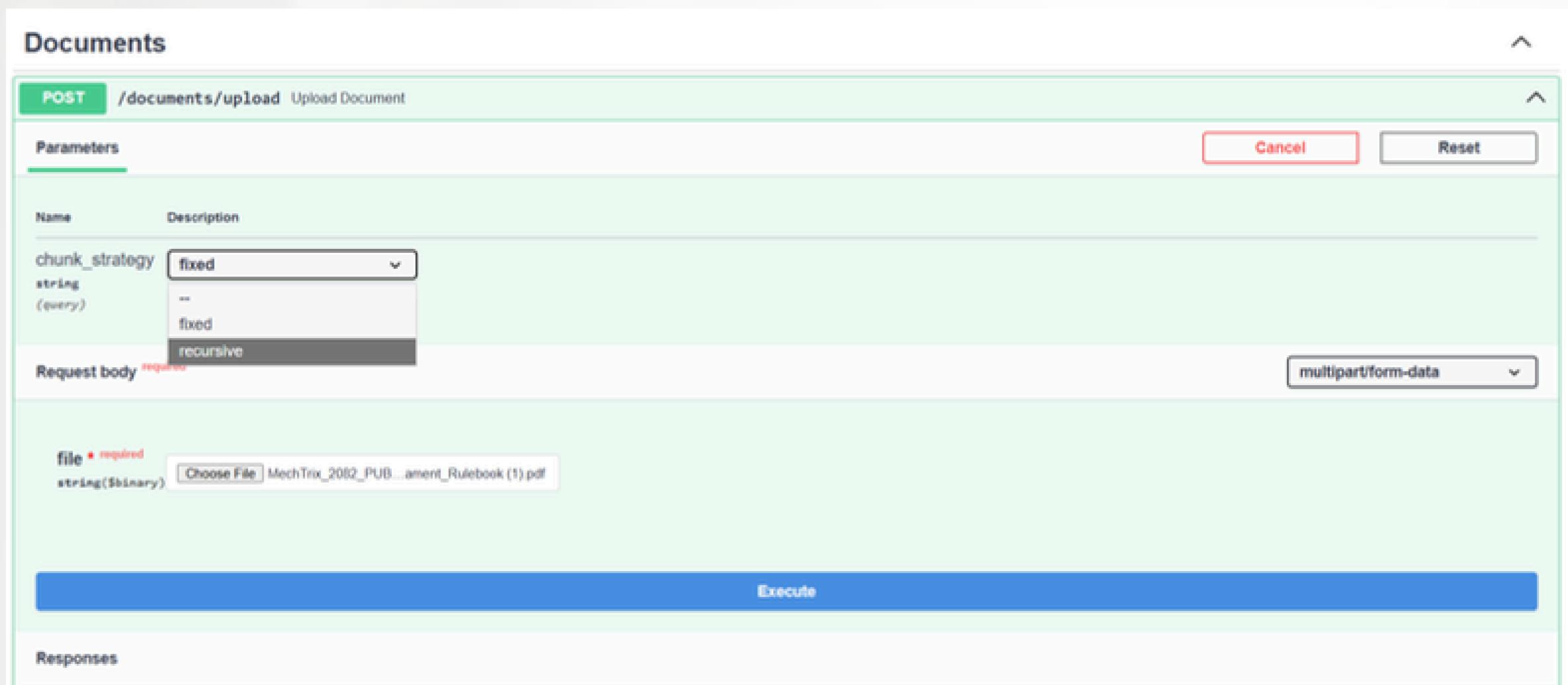
Name	Description
chunk_strategy	fixed -- recursive

Request body required

file required
string(\$binary)
Choose File MechTrix_2002_PUB...ament_Rulebook (1).pdf

Execute

Responses



System Workflow

In This phase:

- Documents are uploaded through the API endpoint in the form of PDF or TXT files
PDF files are converted into text using pdfplumber.
- The system supports two selectable chunking strategies:
 - Fixed-size chunking
 - Recursive chunking
- After chunking, the text chunks are converted into embeddings using a HuggingFace Sentence Transformer model.
- The generated embeddings are stored in Qdrant, which runs as a Docker container to ensure a consistent and isolated environment.
- All document-related metadata (such as document ID, filename, and chunk references) is stored in PostgreSQL for structured and persistent storage.

System Workflow

strategy	id	num_chunks	created_at	filename	file_type	chunk_
	9aad79ae-9292-43dd-97f7-19eab3d8ee31	43	2026-01-19 04:42:52.248658	MechTrix_2082_PUBG_Mobile_Tournament_Rulebook (1).pdf	application/pdf	fixed

(1 row)

Document metadata is stored in PostgreSQL, including details such as document ID, filename, and chunk references, ensuring structured and persistent storage.

```
for idx, (chunk_text, vec) in enumerate(zip(chunks, vector)):
    points.append(
        models.PointStruct(
            id=str(uuid.uuid4()),
            vector=vec.tolist() if hasattr(vec, "tolist") else vec,
            payload={
                "text": chunk_text,
                "filename": filename,
                "chunk_id": idx
            }
        )
    )
```

Each embedded chunk is stored in Qdrant as a point containing the vector representation and payload.

System Workflow

Chat

POST /chat/ask Chat With Pdf

Parameters

No parameters

Request body required

Edit Value | Schema

```
{  
  "query": "String",  
  "session_id": "default_user"  
}
```

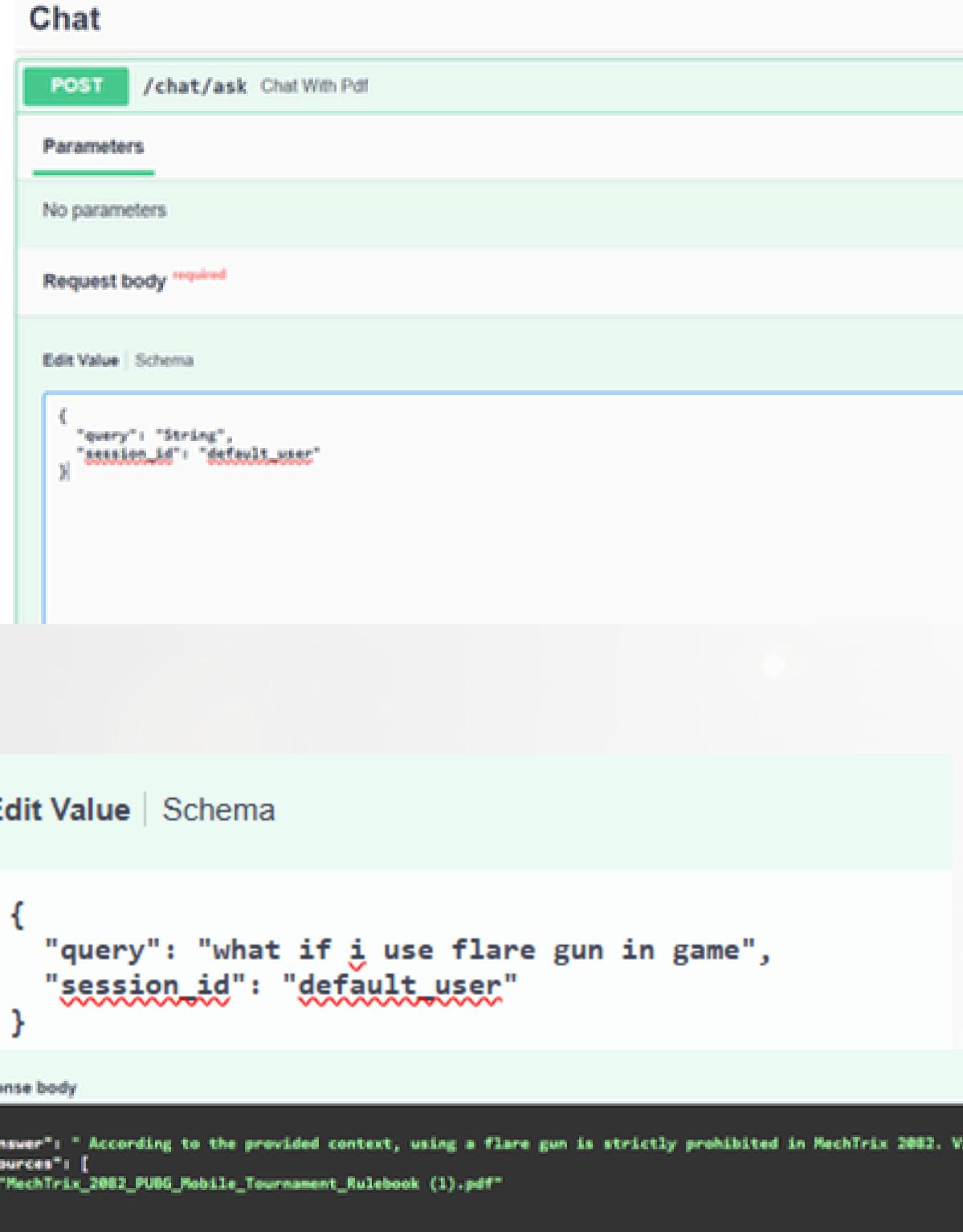
Edit Value | Schema

```
{  
  "query": "what if i use flare gun in game",  
  "session_id": "default_user"  
}
```

response body

```
{  
  "answer": "According to the provided context, using a flare gun is strictly prohibited in MechTrix 2002. Violating these rules may lead to penalties or disqualification.",  
  "sources": [  
    "MechTrix_2002_PUBG_Mobile_Tournament_Rulebook_(1).pdf"  
  ]  
}
```

 Download



User queries are handled through the Chat API endpoint.

Each request is associated with a session_id, which is used to store and retrieve conversation history from Redis, enabling multi-turn conversational support.

User queries are processed through the chat API endpoint, and each interaction is associated with a session ID that is used to store

System Workflow

```
-d "{}\nquery": "My name is navin, navin@gmail.com, 2026-03-03,07:30",\n"session_id": "name"\n"}  
request URL:  
http://127.0.0.1:8000/chat/ask  
server response  
code Details  
200 Response body  
{  
  "answer": "To schedule your interview on March 3rd, 2026 at 07:30, kindly confirm the availability of this slot. If it's suitable for you, I will proceed to book it for you.\nBest Regards,\nHR Assistant [University, Kathmandu]\nYour interview has been successfully recorded in our database.",  
}  
Content-Type: application/json  
Date: Mon, 20 Feb 2026 04:32:37 GMT  
Server: WSGI/1.0  
Content-Length: 240
```

Interviews are booked through the conversational chat flow, where required details such as name, email, date, and time are collected across multiple interactions and stored persistently in PostgreSQL.

```
rag_DB=# SELECT * FROM interview_bookings;  
 id | candidate_name | candidate_email | interview_date | interview_time | created_at  
---+-----+-----+-----+-----+-----  
 6 | Sulav | sual@gmail.com | 2026-02-10 | 10:50:00 | 2026-01-20 04:32:37.856338  
 7 | navin | navin@gmail.com | 2026-03-03 | 07:30:00 | 2026-01-20 05:01:27.386307  
(2 rows)
```

System Workflow

```
127.0.0.1:6379> KEYS *
1) "chat:name"
2) "chat:name_12"
3) "chat:name_1"
4) "chat:suman_chhetri"
5) "chat:name1234"
6) "chat:default_user"
7) "chat:name222"
127.0.0.1:6379> LRANGE "chat:name1234" 0 -1
1) {"role": "Assistant", "content": "Your preferred date for the interview is September 1st, 2082, at 2:00 PM. However, I don't have any information about your chosen room or portfolio details. Could you please provide those as well to complete the booking process?"}
2) {"role": "User", "content": "my name is AJAY and ajay@gmail.com,2082-09-1,2:00,"}
3) {"role": "Assistant", "content": "To proceed with the booking of an interview, I'll need some essential details:\n- Name (Full name)\n- Email address\n- Preferred date (in YYYY-MM-DD format)\n- Time in HH:MM format\nOnce you provide these details, I can confirm your interview booking."}
4) {"role": "User", "content": "ok want ot book interview"}
5) {"role": "Assistant", "content": "The third rule of the MechTrix 2082 PUBG Mobile Tournament is not specified in the provided context. However, it generally pertains to the topic of \\"portsmanship,\\" which refers to fair play, good behavior, and respect towards other participants and the Tournament as a whole."}
6) {"role": "User", "content": "what is rule 3 of tournament"}
127.0.0.1:6379>
```

Redis is used to manage conversation memory by creating a unique key for each user session and storing chat messages in a list structure, enabling efficient multi-turn conversations for multiple users.

System Workflow

Due to API quota limitations with Gemini, I switched to a local Mistral model using Ollama without changing the RAG pipeline.

Conclusion:

This project demonstrates the design and implementation of a robust backend system for document ingestion and conversational retrieval using FastAPI. The solution supports structured document processing, semantic search through a custom RAG pipeline, and multi-turn conversational interactions with reliable context management.

By integrating Qdrant for vector storage, Redis for conversation memory, PostgreSQL for persistent metadata and interview booking storage, and a local Mistral language model via Ollama, the system achieves scalability, modularity, and cost efficiency. The architecture follows clean coding practices and adheres strictly to the defined constraints, avoiding external retrieval frameworks and UI dependencies.

Overall, the project showcases practical backend engineering skills, effective use of modern AI infrastructure, and the ability to build production-oriented systems that balance performance, reliability, and maintainability.