STUDY BOT

AI-Powered Context-Aware Learning Assistant

Developed Using:
FastAPI | MongoDB Atlas | LangChain | Groq LLM | Render Hosting

Submitted By:
Harshitha B.G.

Date:
20-02-2026

PROJECT OVERVIEW AND MEMORY IMPLEMENTATION

1. Project Overview

1.1 Introduction

The Study Bot is an AI-powered chatbot designed to assist students with academic queries while maintaining contextual awareness across multiple interactions. Traditional chatbots process each request independently, without remembering previous conversations. In contrast, Study Bot implements persistent memory storage to provide coherent and personalized responses.

The system is developed using:

- **FastAPI** for backend API development
- **MongoDB Atlas** for persistent chat memory
- **LangChain** for structured prompt management
- **Groq LLM (openai/gpt-oss-20b)** for AI response generation
- **Render** for hosting the deployed API

The chatbot is deployed as a RESTful API, allowing seamless integration with web and mobile applications.

1.2 Objectives

The primary objectives of the Study Bot project are:

- To develop an AI-based academic assistant
- To implement contextual conversation handling
- To store and retrieve chat history using a cloud database
- To provide multi-session user-specific interactions
- To design a scalable and production-ready API system

1.3 System Architecture Overview

The system follows a structured workflow:

1. The user sends a question through an API request.
2. The FastAPI backend receives the request.
3. The system retrieves previous chat history from MongoDB using the user ID.
4. The retrieved history is injected into a structured prompt template.
5. The prompt is sent to the Groq Large Language Model.
6. The model generates a contextual response.
7. Both the user query and the AI response are stored in MongoDB.
8. The response is returned to the user.

This architecture ensures contextual continuity and scalable deployment.

2. Memory Implementation

2.1 Need for Memory

Large Language Models (LLMs) are stateless by default, meaning they do not remember previous interactions unless past conversation data is explicitly included in each request.

To overcome this limitation, Study Bot implements a database-backed memory system. This allows the chatbot to:

- Maintain conversation continuity
- Provide relevant follow-up responses
- Support multi-session learning
- Personalize responses based on user history

2.2 Types of Memory Used

2.2.1 Short-Term Memory

Short-term memory refers to recent conversation history retrieved before generating a response.

- The system retrieves the latest six messages for a specific user.
- Messages are sorted chronologically.
- These messages are injected into the prompt template.

This ensures the model understands the immediate conversational context.

2.2.2 Long-Term Memory

Long-term memory is implemented using MongoDB Atlas.

- Every user message and assistant response is stored permanently.
- Conversations are associated with a unique user ID.
- Each message is timestamped for chronological retrieval.

This allows the chatbot to remember previous interactions even after sessions end.

2.3 Database Structure

Database Name: Chat
Collection Name: users

Each message is stored as a document containing:

- user_id – Identifies the user
- role – Specifies whether the message is from the user or assistant
- message – Stores the conversation text
- timestamp – Records the time of message creation

This structure enables efficient filtering, sorting, and retrieval of user-specific conversation history.

2.4 Memory Retrieval Process

The memory retrieval function performs the following operations:

1. Filters records using the provided user ID.
2. Sorts messages by timestamp in chronological order.
3. Limits results to the most recent messages.
4. Formats them into role-message pairs.
5. Injects them into the prompt template.

This structured retrieval ensures relevant and coherent context injection before generating a response.

2.5 Context Injection Mechanism

The prompt template consists of:

- A system instruction defining chatbot behavior
- A placeholder for conversation history
- The current user question

When a new query is received:

1. Previous messages are fetched from MongoDB.
2. These messages are inserted into the history placeholder.
3. The complete structured prompt is sent to the LLM.

This allows the model to generate responses that reference earlier discussions, making the chatbot context-aware.

2.6 Example Scenario

First Interaction:
User: Explain Newton's Laws
Bot: Provides explanation and stores it in the database.

Second Interaction:
User: Give a real-life example of the second law.

System Process:

- Retrieves previous discussion about Newton's Laws.
- Injects it into the prompt.
- Generates a relevant example response.

This demonstrates how memory enables continuous and meaningful conversations.

---

3. Conclusion

The Study Bot integrates **AI response generation with persistent memory** to create a context-aware academic assistant. By combining **FastAPI**, **MongoDB Atlas**, **LangChain**, **Groq LLM**, and **Render hosting**, the system supports:

- Structured prompt handling
- Scalable API deployment

The memory architecture ensures continuity, efficient storage, and personalized learning support.

4. Project Deployment and Deliverables

The following components are included as part of the project submission:

4.1 GitHub Repository

A complete GitHub repository containing:

- Backend Python file - **app.py**
- Configuration file - **.gitignore**
- Requirements file - **requirements.txt**
- Documentation - **Project_Documentation.txt**

GitHub Repository Link:

Study Bot Github Repository URL - https://github.com/harshithabg333/Study-Bot

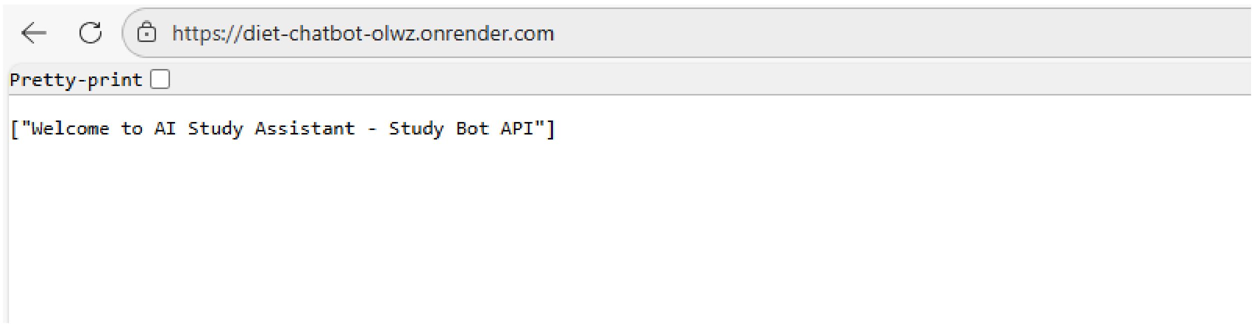
---

4.2 Hosted API (Render)

The Study Bot API is deployed on Render:

Hosted API URL:

Hosted API URL:
https://diet-chatbot-olwz.onrender.com



```
["Welcome to AI Study Assistant - Study Bot API"]
```

1. **Screenshot of hosted API in Render**

Note: This URL includes a random suffix because the project is deployed on Render Free Plan. Free plan does not allow custom subdomains. For professional deployment or a clean URL, a paid plan or custom domain is required.

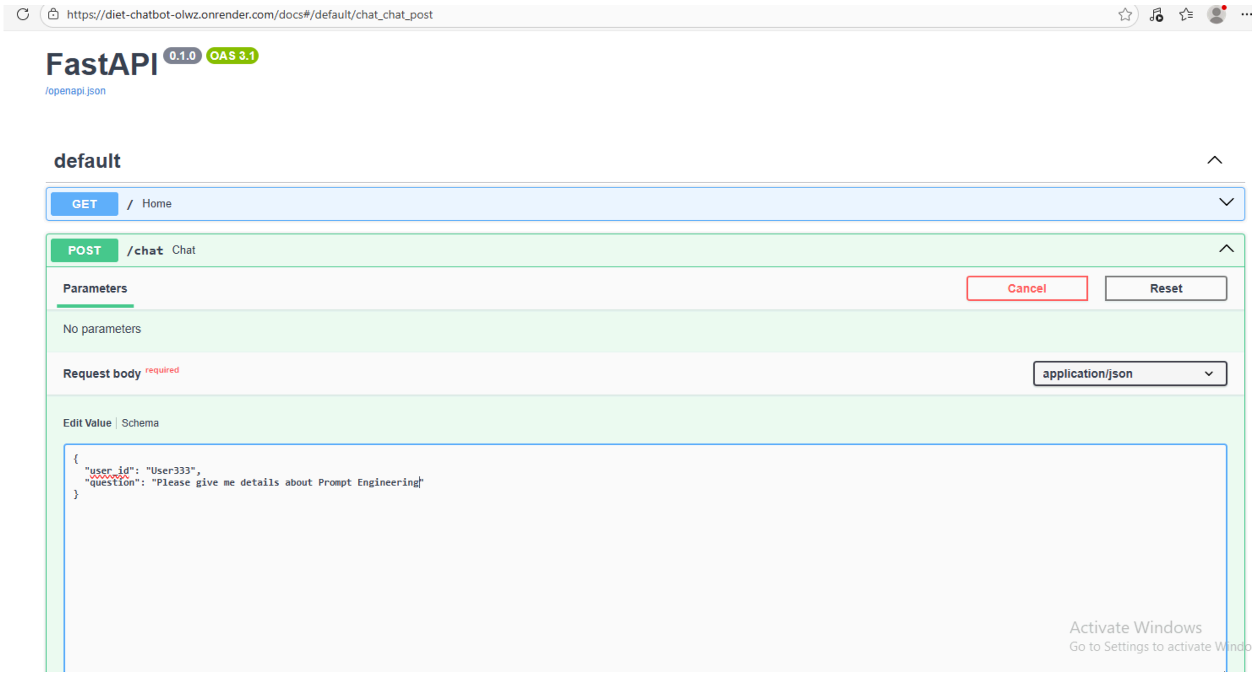
4.3 Sample API Test Screenshots

The submission includes screenshots demonstrating:

- Successful API request using Render -

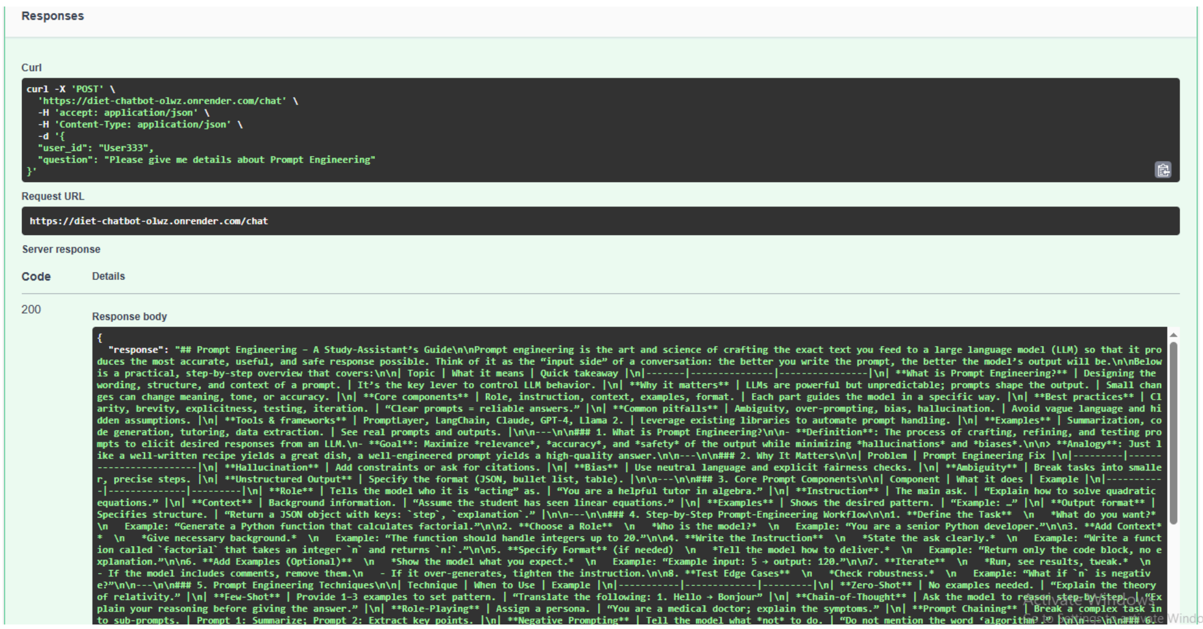
  https://diet-chatbot-olwz.onrender.com/docs

- Request body example



```
{
  "user_id": "User333",
  "question": "Please give me details about Prompt Engineering"
}
```

2. **Screenshot of Request**



Responses

Curl

```
curl -X 'POST' \
  'https://diet-chatbot-olwz.onrender.com/chat' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "user_id": "User333",
  "question": "Please give me details about Prompt Engineering"
}'
```

Request URL

```
https://diet-chatbot-olwz.onrender.com/chat
```

Server response

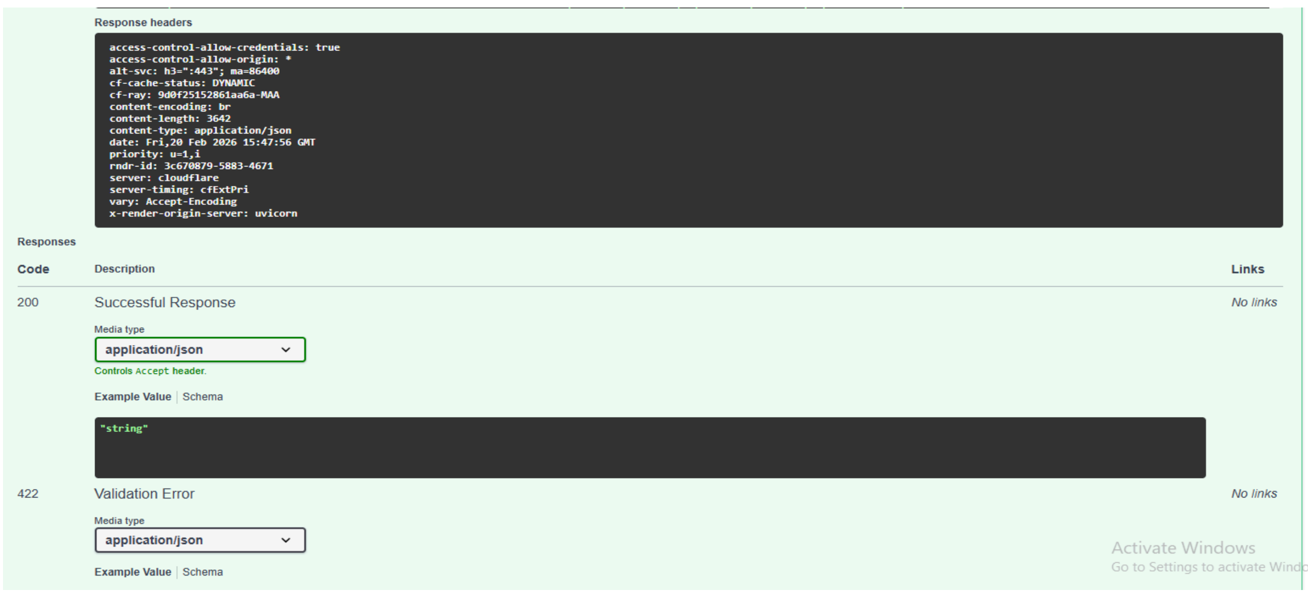
Code Details

200

Response body

```
{
  "response": "## Prompt Engineering – A Study-Assistant’s Guide\n\nPrompt engineering is the art and science of crafting the exact text you feed to a large language model (LLM) so that it pro
duces the most accurate, useful, and safe response possible. Think of it as the “input side” of a conversation: the better you write the prompt, the better the model’s output will be.\n\nBelow
is a practical, step-by-step overview that covers:\n\n| Topic | What it means | Quick takeaway |\n|-------|---------------|----------------|\n| **What is Prompt Engineering?** | Designing the
wording, structure, and context of a prompt. | It’s the key lever to control LLM behavior. |\n| **Why it matters** | LLMs are powerful but unpredictable; prompts shape the output. | Small chan
ges can change meaning, tone, or accuracy. |\n| **Core components** | Role, instruction, context, examples, format. | Each part guides the model in a specific way. |\n| **Best practices** | Cl
arity, brevity, explicitness, testing, iteration. | “Clear prompts = reliable answers.” |\n| **Common pitfalls** | Ambiguity, over-prompting, bias, hallucination. | Avoid vague language and hi
dden assumptions. |\n| **Tools & frameworks** | PromptLayer, LangChain, Claude, GPT-4, Llama 2. | Leverage existing libraries to automate prompt handling. |\n| **Examples** | Summarization, co
de generation, tutoring, data extraction. | See real prompts and outputs. |\n\n---\n\n### 1. What is Prompt Engineering?\n\n- **Definition**: The process of crafting, refining, and testing pro
mpts to elicit desired responses from an LLM.\n- **Goal**: Maximize *relevance*, *accuracy*, and *safety* of the output while minimizing *hallucinations* and *biases*.\n\n> **Analogy**: Just l
ike a well-written recipe yields a great dish, a well-engineered prompt yields a high-quality answer.\n\n---\n\n### 2. Why It Matters\n\n| Problem | Prompt Engineering Fix |\n|---------|------
------------------|\n| **Hallucination** | Add constraints or ask for citations. |\n| **Bias** | Use neutral language and explicit fairness checks. |\n| **Ambiguity** | Break tasks into smalle
r, precise steps. |\n| **Unstructured Output** | Specify the format (JSON, bullet list, table). |\n\n---\n\n### 3. Core Prompt Components\n\n| Component | What it does | Example |\n|---------
-|-------------|---------|\n| **Role** | Tells the model who it is “acting” as. | “You are a helpful tutor in algebra.” |\n| **Instruction** | The main ask. | “Explain how to solve quadratic
equations.” |\n| **Context** | Background information. | “Assume the student has seen linear equations.” |\n| **Examples** | Shows the desired pattern. | “Example: …” |\n| **Output format** |
Specifies structure. | “Return a JSON object with keys: `step`, `explanation`.” |\n\n---\n\n### 4. Step-by-Step Prompt-Engineering Workflow\n\n1. **Define the Task**  \n   *What do you want?*
\n   Example: “Generate a Python function that calculates factorial.”\n\n2. **Choose a Role**  \n   *Who is the model?*  \n   Example: “You are a senior Python developer.”\n\n3. **Add Context*
*  \n   *Give necessary background.*  \n   Example: “The function should handle integers up to 20.”\n\n4. **Write the Instruction**  \n   *State the ask clearly.*  \n   Example: “Write a funct
ion called `factorial` that takes an integer `n` and returns `n!`.”\n\n5. **Specify Format** (if needed)  \n   *Tell the model how to deliver.*  \n   Example: “Return only the code block, no e
xplanation.”\n\n6. **Add Examples (Optional)**  \n   *Show the model what you expect.*  \n   Example: “Example input: 5 → output: 120.”\n\n7. **Iterate**  \n   *Run, see results, tweak.*  \n
- If the model includes comments, remove them.\n   - If it over-generates, tighten the instruction.\n\n8. **Test Edge Cases**  \n   *Check robustness.*  \n   Example: “What if `n` is negativ
e?”\n\n---\n\n### 5. Prompt Engineering Techniques\n\n| Technique | When to Use | Example |\n|-----------|-------------|---------|\n| **Zero-Shot** | No examples needed. | “Explain the theory
of relativity.” |\n| **Few-Shot** | Provide 1-3 examples to set pattern. | “Translate the following: 1. Hello → Bonjour” |\n| **Chain-of-Thought** | Ask the model to reason step-by-step. | “Ex
plain your reasoning before giving the answer.” |\n| **Role-Playing** | Assign a persona. | “You are a medical doctor; explain the symptoms.” |\n| **Prompt Chaining** | Break a complex task in
to sub-prompts. | Prompt 1: Summarize; Prompt 2: Extract key points. |\n| **Negative Prompting** | Tell the model what *not* to do. | “Do not mention the word ‘algorithm’.” |\n\n---\n\n### 6.
```

3. Screenshot of the Response body



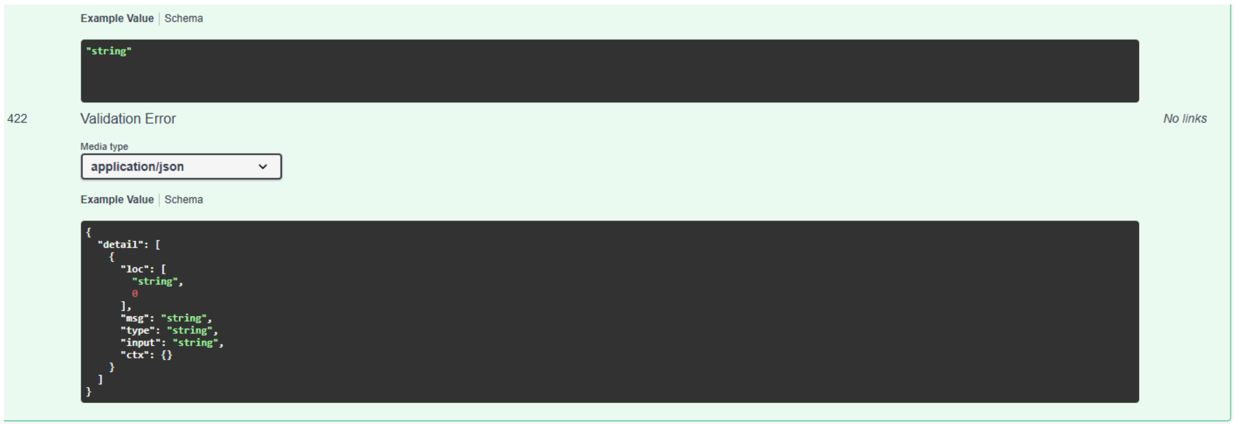
Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
alt-svc: h3=":443"; ma=86400
cf-cache-status: DYNAMIC
cf-ray: 9d0f25152861aa6a-MAA
content-encoding: br
content-length: 3642
content-type: application/json
date: Fri,20 Feb 2026 15:47:56 GMT
priority: u=1,i
rndr-id: 3c670879-5883-4671
server: cloudflare
server-timing: cfExtPri
vary: Accept-Encoding
x-render-origin-server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls Accept header. Example Value | Schema "string"	No links
422	Validation Error Media type: application/json Example Value | Schema	No links

4.Screenshot of Response Header


Example Value | Schema

```
"string"
```

422 Validation Error

No links

Media type: application/json

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string",
      "input": "string",
      "ctx": {}
    }
  ]
}
```

5. Screenshot of Validation error id any

Thank You 😀