Study Bot Project

AI-Powered Study Assistant with Conversation Memory

Project Type:	Chatbot with Database Memory
Technology Stack:	FastAPI, MongoDB, Groq LLM, Langchain
Deployment Platform:	Render
Submission Date:	February 21, 2026

Table of Contents

1. Project Overview

The Study Bot is an AI-powered chatbot designed to assist students with their academic queries. It leverages advanced language models to provide intelligent, context-aware responses while maintaining conversation history through MongoDB database integration.

Key Objectives:

• Provide instant, accurate answers to study-related questions

• Maintain conversation context across multiple interactions

• Store and retrieve chat history for personalized learning experiences

• Offer a scalable, API-based solution for multi-user support

• Deploy a production-ready application accessible via web

2. Features and Capabilities

AI-Powered Responses: Uses Groq's Llama 3.1 70B model for intelligent, accurate responses to academic questions across various subjects.

Conversation Memory: Stores complete chat history in MongoDB, enabling context-aware conversations and personalized learning experiences.

Multi-User Support: Handles multiple users simultaneously with isolated conversation histories for each user.

Study-Focused: Specialized system prompt optimized for educational content, study tips, and learning strategies.

RESTful API: Clean, well-documented API endpoints for easy integration with other applications.

Real-Time Processing: Fast response times with asynchronous processing capabilities.

3. Technical Architecture

Technology Stack:

Component	Technology	Purpose
Backend Framework	FastAPI	RESTful API development
LLM Provider	Groq	AI language model hosting
LLM Integration	Langchain	LLM orchestration and chaining
Database	MongoDB	Conversation history storage
Server	Uvicorn	ASGI web server
Deployment	Render	Cloud hosting platform

Architecture Flow:

1. **User Request:** Client sends POST request to /chat endpoint with user_id and message

2. **History Retrieval:** System fetches last 10 conversation exchanges from MongoDB

3. **Context Building:** Previous messages are formatted and sent to LLM along with new query

4. **LLM Processing:** Groq processes the request using Llama 3.1 model with system prompt

5. **Response Generation:** LLM generates context-aware response

6. **Storage:** New exchange is stored in MongoDB with timestamp

7. **Response Delivery:** JSON response sent back to client

4. Memory Implementation

The memory system is the core feature that distinguishes this chatbot from stateless alternatives. It enables the bot to maintain context across conversations, providing a more natural and personalized learning experience.

Implementation Details:

Storage Mechanism: Each conversation exchange (user message + bot response) is stored as a document in MongoDB with user_id, timestamp, and message content.

Retrieval Strategy: When processing a new message, the system retrieves the last 10 exchanges for that user, ensuring manageable context length while maintaining relevance.

Context Window: Retrieved messages are formatted as alternating Human/AI messages and sent to the LLM along with the system prompt and new query.

Indexing: MongoDB indexes on user_id and timestamp ensure fast query performance even with large conversation histories.

Efficiency: Limiting to 10 exchanges balances context depth with API token costs and response time.

Benefits of This Approach:

• Students can ask follow-up questions without repeating context

• Bot can reference previous explanations and build upon them

• Personalized learning paths based on user's question history

• Reduced need for users to re-explain their learning needs

5. API Endpoints

`GET /`

Returns API information and list of available endpoints

`GET /health`

Health check endpoint - verifies API and database connectivity

`POST /chat`

Main chat endpoint - accepts user message and returns bot response

`GET /history/{user_id}`

Retrieves conversation history for specified user

`DELETE /clear-history/{user_id}`

Clears all conversation history for specified user

`GET /stats`

Returns system statistics (total users, conversations)

Request/Response Examples:

POST /chat Request:

```
{"user_id": "student_123", "message": "What is photosynthesis?"}
```

POST /chat Response:

```
{"user_id": "student_123", "user_message": "What is photosynthesis?",
"bot_response": "Photosynthesis is the process...", "timestamp":
"2024-02-21T10:30:00"}
```

6. Database Schema

MongoDB Collection: chat_history

Field	Type	Description
_id	ObjectId	Auto-generated unique identifier
user_id	String	User identifier (indexed)
user_message	String	Message sent by user
bot_response	String	Response generated by bot
timestamp	DateTime	UTC timestamp of exchange (indexed)

Indexes:

• user_id - Enables fast filtering by user

• timestamp - Enables efficient chronological ordering

7. Deployment Process

Step 1: MongoDB Atlas Setup

• Created free MongoDB Atlas cluster

• Configured database user with read/write permissions

• Whitelisted IP addresses (0.0.0.0/0 for Render)

• Obtained connection string

Step 2: Code Repository

• Initialized Git repository

• Committed all project files

• Pushed to GitHub

• Ensured .env file is in .gitignore

Step 3: Render Deployment

• Created new Web Service on Render

• Connected GitHub repository

• Configured build command: pip install -r requirements.txt

• Configured start command: uvicorn main:app --host 0.0.0.0 --port $PORT

• Added environment variables (MONGODB_URI, GROQ_API_KEY, DB_NAME)

• Deployed application

Environment Variables:

Variable	Purpose
MONGODB_URI	MongoDB Atlas connection string
GROQ_API_KEY	Groq API authentication key
DB_NAME	Database name (studybot)
PORT	Server port (auto-assigned by Render)

8. Testing and Validation

Comprehensive testing was performed to ensure all components function correctly. A dedicated test script (test_api.py) was developed to validate all API endpoints.

Test Cases:

Health Check: Verified API and database connectivity

Chat Functionality: Tested with various study questions across different subjects

Context Awareness: Verified bot remembers previous conversation

History Retrieval: Confirmed accurate storage and retrieval of chat history

Multi-User Support: Tested isolation of conversations between different users

Clear History: Validated deletion of user conversation history

Statistics: Verified accurate counting of users and conversations

Sample Test Conversation:

User: What is photosynthesis?

Bot: Photosynthesis is the process by which plants convert light energy into chemical energy...

User: Can you explain it in simpler terms?

Bot: Sure! Think of photosynthesis as plants making their own food using sunlight...

User: What do plants need for this process?

Bot: Based on our discussion about photosynthesis, plants need three main things: sunlight, water, and carbon dioxide...

9. Code Structure

Project Files:

File	Purpose	Lines
main.py	FastAPI application and endpoints	~170
chatbot.py	LLM integration and response generation	~100
database.py	MongoDB connection and operations	~140
requirements.txt	Python dependencies	~7
test_api.py	API testing script	~100
.env.example	Environment variables template	~8
README.md	Documentation	~400
.gitignore	Git ignore rules	~40

Key Components:

StudyBot Class: Manages LLM interactions, formats prompts, and handles conversation context

ChatDatabase Class: Handles all MongoDB operations including storage, retrieval, and queries

FastAPI Routes: Defines API endpoints with request/response validation

Pydantic Models: Ensures type safety and automatic validation for API requests/responses

10. Future Enhancements

The current implementation provides a solid foundation for an AI study assistant. The following enhancements could further improve functionality and user experience:

User Authentication

- Implement JWT-based authentication
- Add user registration and login
- Secure API endpoints with token verification

Advanced Memory Management

- Implement conversation summarization for very long histories
- Add semantic search across previous conversations
- Enable topic-based conversation threading

Enhanced Study Features

- Generate practice questions based on conversation history
- Create flashcards from key concepts discussed
- Provide spaced repetition reminders
- Track learning progress and topics covered

Multi-Modal Capabilities

- Support image uploads for diagram explanations
- Handle PDF document analysis
- Generate visual study aids

Performance Optimization

- Implement caching for common queries
- Add rate limiting to prevent abuse

- Optimize database queries with aggregation pipelines

Analytics Dashboard

- Track most common questions by subject
- Monitor bot performance and accuracy
- Visualize user engagement metrics

Conclusion

The Study Bot project successfully demonstrates the implementation of an AI-powered chatbot with persistent memory using modern web technologies. The application combines the power of large language models with efficient data storage to create a practical tool for student learning.

Key achievements include successful integration of Groq's LLM API, implementation of context-aware conversations through MongoDB storage, deployment of a production-ready API on Render, and comprehensive testing to ensure reliability. The project meets all specified requirements and provides a strong foundation for future enhancements.

Project Requirements Checklist:

✓ FastAPI application with RESTful endpoints

✓ LLM integration using Groq and Langchain

✓ MongoDB database for chat history storage

✓ Context-aware responses using memory

✓ Study-focused system prompt

✓ Multi-user support with isolated histories

✓ Deployed on Render with proper configuration

✓ Comprehensive documentation (README)

✓ Testing suite with validation scripts

✓ GitHub repository with clean code structure

This project demonstrates proficiency in full-stack development, API design, database management, and cloud deployment - essential skills for modern software engineering.