Zep Memory Chatbot with Graph Visualization

This document provides a detailed technical explanation of the Zep Memory Chatbot project. The system integrates a Streamlit-based chatbot with Zep for long-term memory, LLM-based reasoning, and graph visualization using st_link_analysis.

1. Project Overview

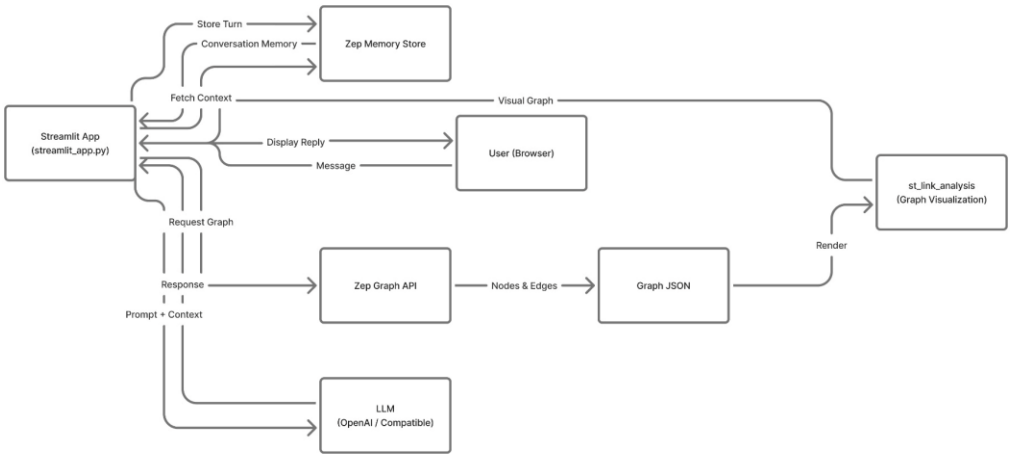
The project is designed to demonstrate how conversational memory can be persisted, retrieved, and visualized using Zep. The chatbot stores each conversation turn in Zep, retrieves relevant context for future responses, and exposes the internal memory graph as an interactive visualization within the Streamlit UI.

2. Core Components

- Streamlit App (streamlit_app.py): User interface, orchestration layer
- Zep Memory Store: Persistent conversation memory and graph backend
- LLM (OpenAI or compatible): Response generation
- Zep Graph API: Node and edge retrieval
- st_link_analysis: Graph visualization component

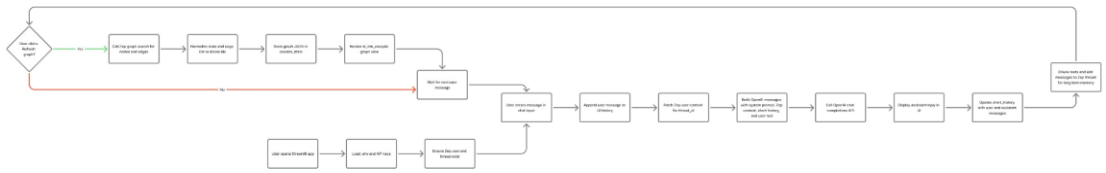
3. System-Level Architecture

The following diagram shows the high-level system architecture and data flow.



4. Streamlit Chatbot Flow

This flow represents the lifecycle of a single chat interaction within the Streamlit application, including context retrieval, LLM invocation, and memory persistence.

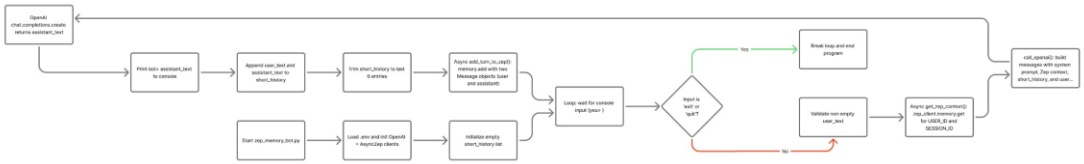


5. Zep Graph Exporter Flow

This flow describes how the application retrieves nodes and edges from the Zep Graph API, normalizes IDs, deduplicates data, and converts it into the JSON format required by st_link_analysis.

6. Zep CLI Memory Bot Flow

This diagram documents the optional CLI-based chatbot implementation. It demonstrates the same memory principles as the Streamlit app but without a UI layer.



7. Data Flow Summary

1. User sends a message via UI or CLI
2. Application retrieves relevant context from Zep
3. Context + user input are sent to the LLM
4. LLM response is returned to the user
5. User and assistant messages are stored back into Zep
6. Graph data can be fetched and visualized at any time

8. Design Rationale

- Zep is used as a centralized memory layer instead of session-based storage
- LLM remains stateless, improving scalability
- Graph visualization reflects real stored memory, not synthetic data
- Streamlit is used only as a presentation and orchestration layer

9. Security and Best Practices

- API keys are loaded from environment variables
- .env files are excluded from version control

• Runtime state files (thread IDs) are not committed
• Secrets are never hardcoded

10. Conclusion

This project demonstrates a production-grade pattern for building memory-enabled AI systems with transparent internal state and visualization. The architecture is extensible, multi-user ready, and suitable for portfolio or enterprise proof-of-concept use.