# Knowledge Graph Generator

## I. Introduction

The primary goal of this application is to leverage a Large Language Model (LLM) for the creation, storage, and visualization of Knowledge Graphs (KGs). This involves extracting meaningful information from LLM responses to construct a structured graph that represents entities, their relationships, and associated attributes.

### A. Conversation Interface

Create an intuitive interface to accept user queries or arbitrary information for processing by the LLM. Design the interface to facilitate a seamless interaction, allowing users to input questions or topics they want information about.

### B. Parsing and Understanding

Begin by parsing the responses obtained from the LLM to extract relevant information. Use the LLM model provided by OpenAI to understand the context, identify entities, and discern relationships from the textual data.

### C. Relationship Identification

Identify relationships between entities based on the context provided in the LLM responses. Consider the nuances of language to accurately represent the connections between different entities.

### D. Knowledge Graph Creation

Organize the extracted entities, relationships, and attributes into a structured graph format.

### E. Integration Mechanisms

Develop mechanisms to integrate data seamlessly into Neo4j, ensuring efficient storage and retrieval operations. This lays the foundation for the initial stages of building an application that creates, stores, and visualizes Knowledge Graphs using an LLM.

## System Architecture

### 1. Application Subsystem:

- Frontend
  Manages user queries and displays the Knowledge Graph. Optionally updates the graph.
- Backend
  Processes user queries, integrates with Neo4j for Knowledge Graph management, and interacts with OpenAI's LLM for natural language understanding.

### 2. Neo4j Subsystem:

- Database Management
  Neo4j stores, retrieves, updates, and manages the structured Knowledge Graph.

3. **OpenAI's LLM Subsystem:**
    - API Integration
      Handles user queries, acts as the data source for the user's prompt and provides responses.
    - KG Updates
      Continuously update the knowledge graph according to the user's input.

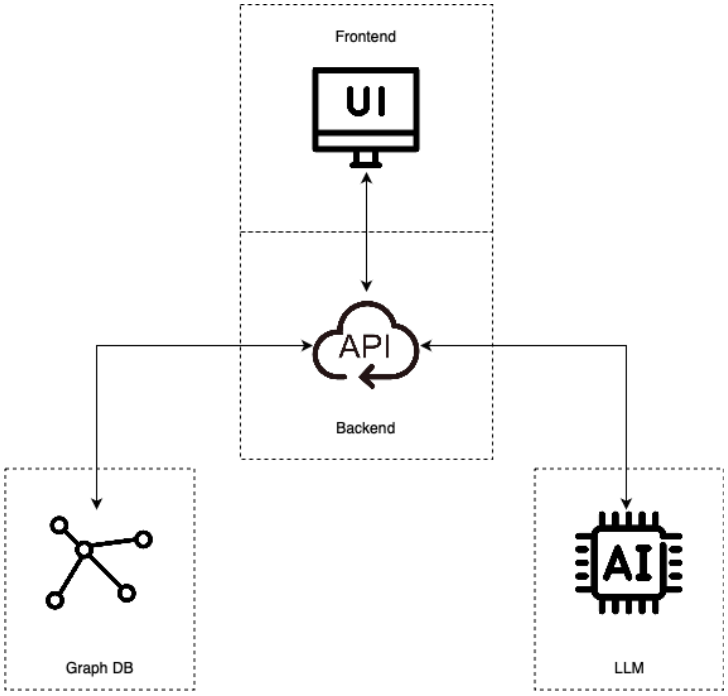


**Fig. 1 System Architecture**

## II. Assumptions

- A maximum limit of 150 nodes in the Knowledge Graph, considering constraints on computational resources, performance, and user experience.
- The application assumes that user queries and prompts are provided in a textual format, and the natural language understanding capabilities of the LLM are optimized for such input.
- The application assumes consistent internet connectivity for accessing OpenAI's LLM and any external data sources required.
- The effectiveness of the Named Entity Recognition (NER) assumes that users provide queries and prompts with a reasonable level of clarity and coherence.
- The application assumes a single user interacting with the system.
- Users will use a monitor with at least 720p resolution when accessing it.

## III. System Interaction

- Frontend communicates with the Backend to submit queries to get text data and visualize the Knowledge Graph.
- Backend interacts with Neo4j for data storage, update and retrieval.
- Backend communicates with OpenAI's LLM for natural language understanding and acts as the data source for the user's prompt.

## IV. Challenges, Considerations, and Suggestions

While some features may not be feasible due to time constraints, the following points highlight challenges, considerations, and specific suggestions for improvement:

- Minimizing latency to improve user responsiveness.
- Ensuring consistent data representation across Neo4j and LLM responses.
- Enhancing scalability for concurrent user access by incorporating multiple graphs and databases.
- Optimizing prompts to improve Named Entity Recognition (NER) accuracy.
- Utilizing data science algorithms like community, similarity, and centrality detection to group and cluster nodes.
- Optimizing the graph by removing redundant nodes and edges.