By : Jason

Github: https://github.com/jasonoesin/knowledge-constellation/

System Architecture:

1. **Frontend (Next.js):**

Next.js Application

- Next.js is a React framework used for building modern web applications.
- It handles the rendering of pages on the client and server side.
- The frontend is responsible for user interface components, interactions, and routing.

2. **Backend (NestJS):**

NestJS Application:

- NestJS is a backend framework that leverages TypeScript and is built on top of Express.js.
- It follows a modular structure using modules, controllers, and services.

3. **Database (Neo4j):**

Neo4j Graph Database:

- Neo4j is a graph database that stores data in nodes and relationships.
- It is particularly well-suited for applications with complex relationships and graph structures.

4. Graph Visualization (D3.JS)

D3.js, reasons I picked it over SigmaJS are:

- D3.js is a highly flexible and powerful library that allows to create a wide range of data visualization types.
- It provides fine-grained control over the elements in the visualization, allowing for deep customization.
- D3.js has a large and active community, which means there are plenty of resources, tutorials, and examples available.

Key Designs and Functionalities

**1. Conversational Interface:**

- A user-friendly web interface where users can input queries.
- Integration with an LLM to process these queries and generate textual responses.

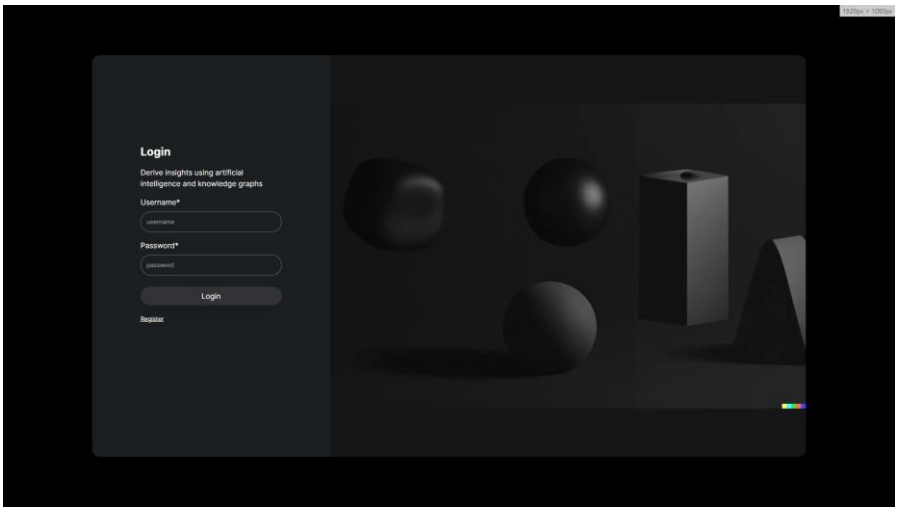


*Figure 1 Register and Login Page*

To implement security measures for data protection and user privacy. I have built the web app using JWT for authentication therefore making it more secure. For scalability, I have made that each user will have each of their own graph.

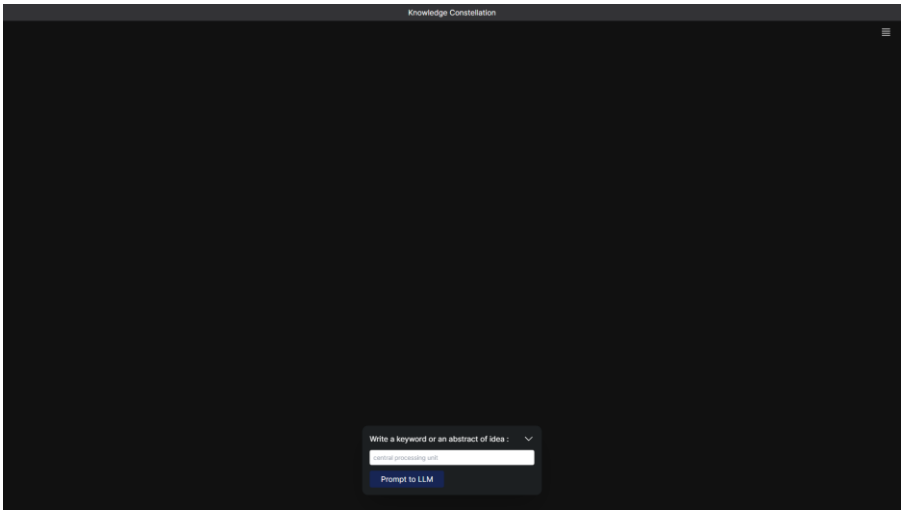


*Figure 2 Empty Graph*

This will be the landing page of each user after logging in. There will be a prompt bar where user can prompt and interface with LLM.

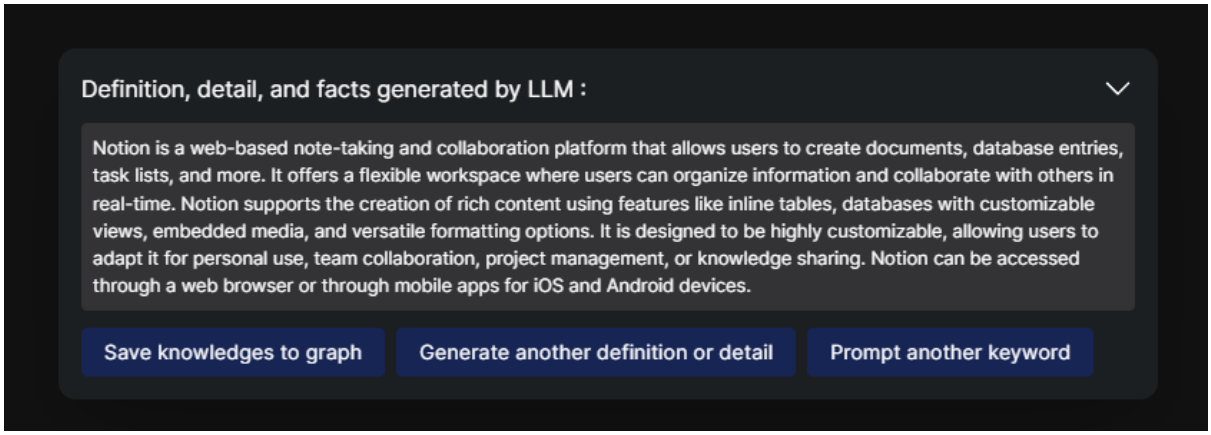


*Figure 3 LLM Returning Response of a definition, detail, and facts*

LLM will be returning a response based on the provided keywords by the user. This response will be converted into knowledge graph by LLM if the user wants to save it to a graph. User can also generate another definition or detail. Also, they can prompt another keyword.

2. Dynamic Knowledge Graph Visualization:

- Utilization of D3.js to visually represent the knowledge graph on the web interface.
- Interactive and dynamic graph that updates in real-time as new information is added.


*Figure 4 Graph after being saved by the user*


*Figure 5 Graph after being updated by new information*

Graph built using D3.JS with interactivity such as dragging, tooltip on the bottom left that explains the nodes, components, and properties. Each node is related by other nodes and have its own relatonship types display on the edges. On the right side there is also an analytics panel where lists all the nodes components categories with its total components. Also, there also lists all the relationships provided on the graph. After updating the graph, graph will dynamically changes based on the new knowledges and informations.

3. Backend Processing with NestJS:

• NestJS backend to handle HTTP requests and manage application logic.

• Convert text to knowledge graph and store it in graph database.

• Integration with the LLM for query processing and obtaining structured information for graph updates.

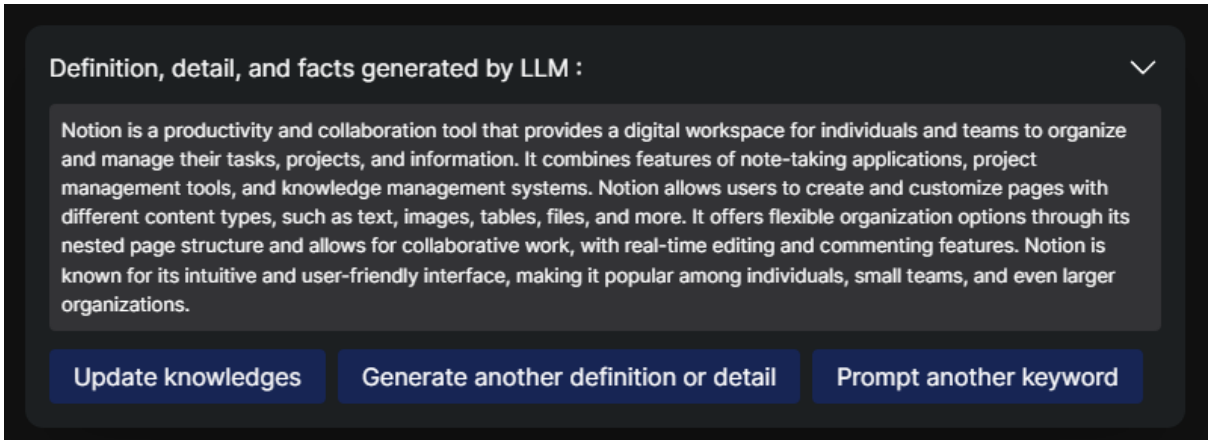


*Figure 6 LLM updating the Neo4J Database by converting the text into knowledge graph in cypher query language.*

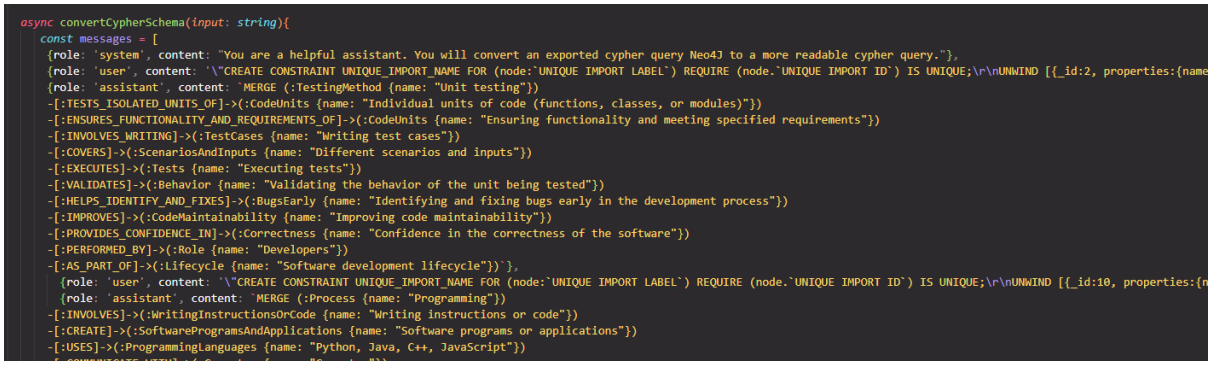


*Figure 7 Example of LLM Optimizing the Schema of Neo4J and Converting it*

4. Neo4j Graph Database:

- Neo4j used for storing and managing the knowledge graph data.
- Dynamic schema capability to allow for the creation of new types of nodes and edges as identified from LLM responses.

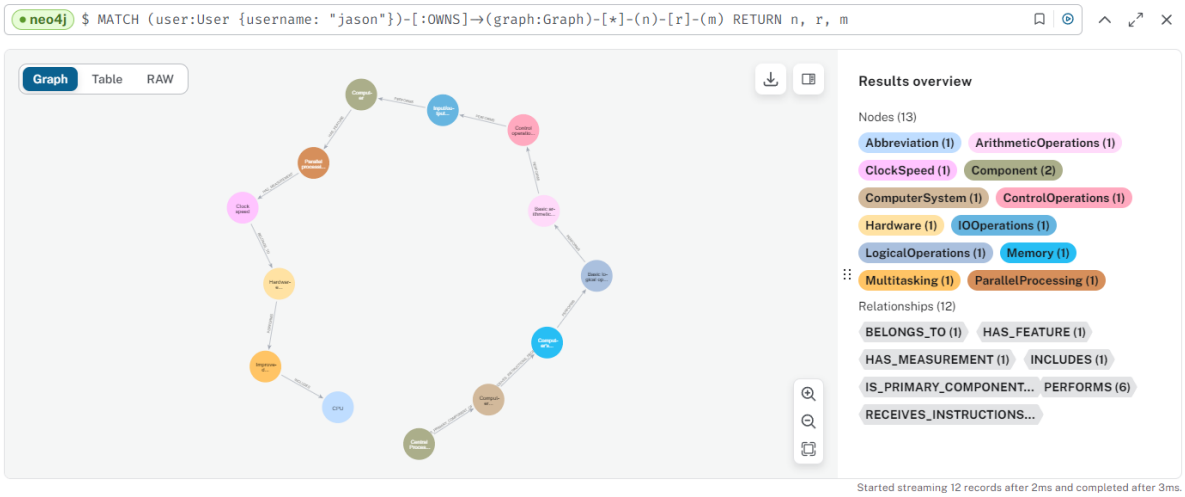


*Figure 8 Neo4j Graph Database (Workspace)*


*Figure 9 Displaying Neo4J Data inside the developed web app*

5. LLM-Assisted Schema Evolution:

- Periodic analysis of the Neo4j graph schema using the LLM to suggest optimizations and refinements.
- Implementing schema updates in Neo4j based on LLM recommendations, such as merging similar node labels or relationship types.
-

6. Frontend Development:

- Implementation of the web interface using HTML, CSS, and JavaScript.
- Integration with Sigma.js for graph visualization, capable of adapting to new types of data dynamically.

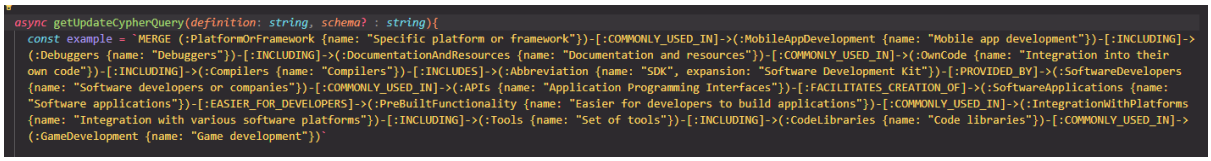


*Figure 10 Getting update cypher query after prompting for new information*

In conclusion, first User will prompt a keyword to the LLM where it will return a response of a detailed text and definition. Then, it will be converted into Cypher Query Language by LLM, and therefore will be run on the Neo4J Database. After that the user will be given a chance to update the graph by prompting other definitions. Then, if the User proceeds to update the graph, LLM will be given two inputs, 1. The current Graph Schema on Converted Cypher Query, 2. The new knowledge graphs from the new definition. It will then optimize and merge into a new Cypher Query Language which will update the graph.