

LLM Retrieval Augmented Generation Application

RAG Customer Support LLM Application

Objective of the practical is to create an RAG Customer Support LLM Chabot application to answer customer questions without Hallucinations.

What are Hallucinations?

LLMs like GPT or other transformer-based models generate text by predicting the next most likely word based on patterns learned from massive text datasets.

However, they do not inherently verify facts — they rely on statistical associations rather than a true understanding of reality.

As a result, when asked a question or given an incomplete prompt, an LLM may:

- Invent facts, numbers, or references that don't exist.
- Confidently present incorrect information in a coherent and authoritative tone.
- Fill in gaps with “best guesses” that are linguistically logical but factually wrong.

What is Retrieval-Augmented Generation (RAG)?

Retrieval-Augmented Generation (RAG) is an advanced technique that enhances Large Language Models (LLMs) by combining **information retrieval** with **text generation**.

Instead of relying solely on the model's internal knowledge, RAG retrieves relevant external documents or data from a **knowledge base or vector database** and uses them to generate more accurate, up-to-date, and context-specific responses.

Key Components

Retriever:

Searches external data sources (e.g., documents, databases, or embeddings) to find the most relevant information for the user query.

Generator (LLM):

Uses the retrieved information to produce a coherent, factual, and context-aware response.

Benefits

Reduces hallucinations by grounding responses in real data

Provides access to domain-specific or private knowledge

Improves factual accuracy and contextual relevance

Example

When a user asks a question about company policies, a RAG system retrieves related documents from the company's internal database and then generates an answer based on that content — ensuring the response is accurate and source-based.

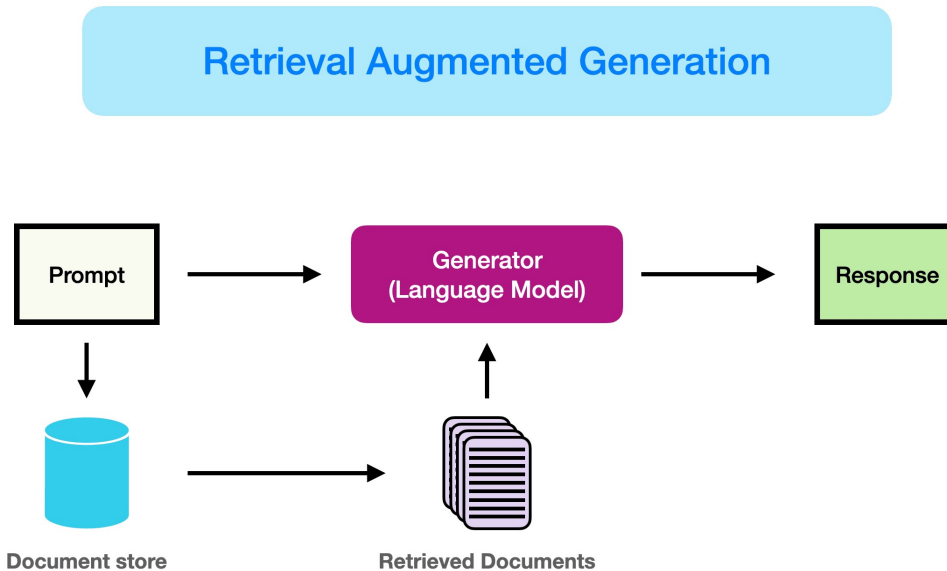


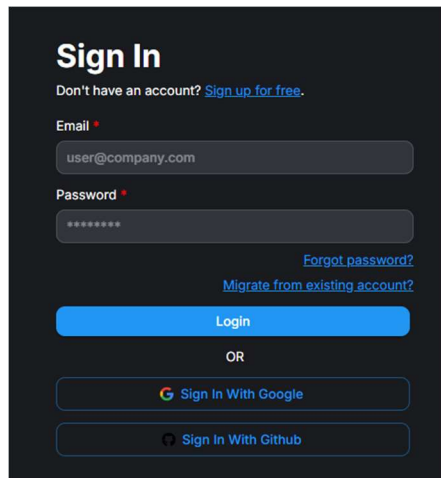
Image source: promptingguide.ai

1. Flowise Website

Goto this Website : <https://cloud.flowiseai.com/signin>

Sign up for free->complete signup of a free Account

Sign In with the new account created



Sign In

Don't have an account? [Sign up for free.](#)

Email *

user@company.com


Password *


[Forgot password?](#)

[Migrate from existing account?](#)

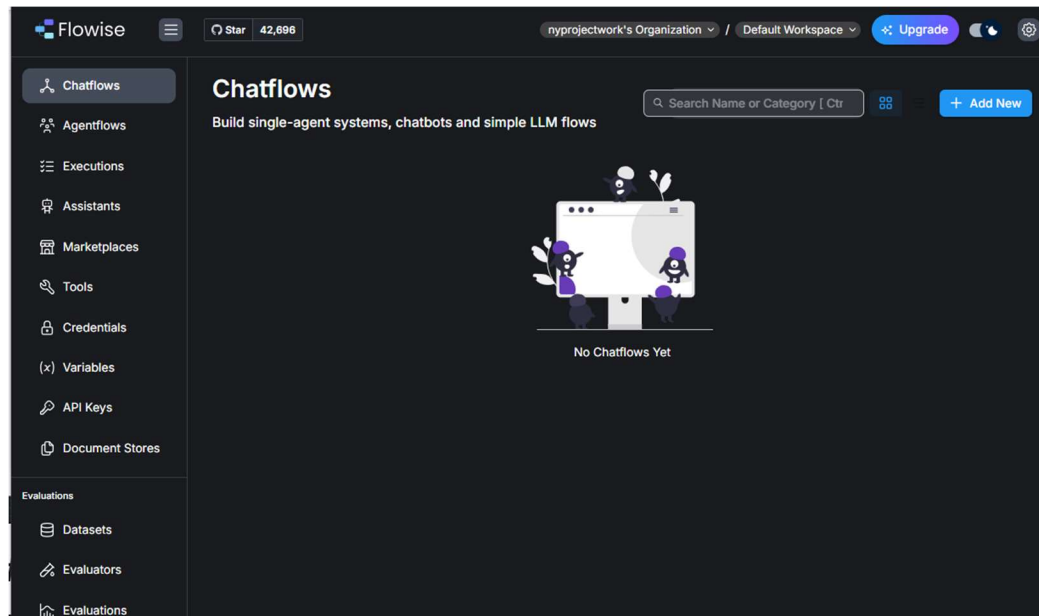
Login

OR

 Sign In With Google

 Sign In With Github

2. Flowise Development Interface



3. Chatflows

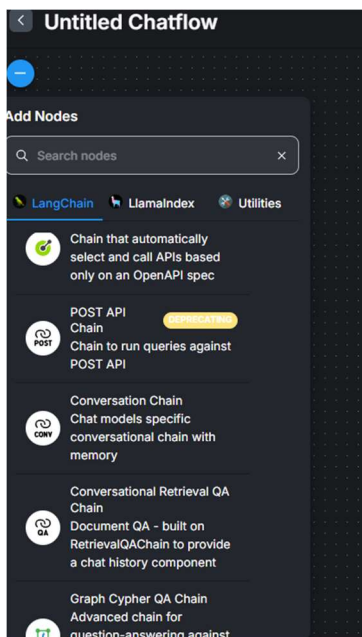
Select Chatflows- > **+Add New**

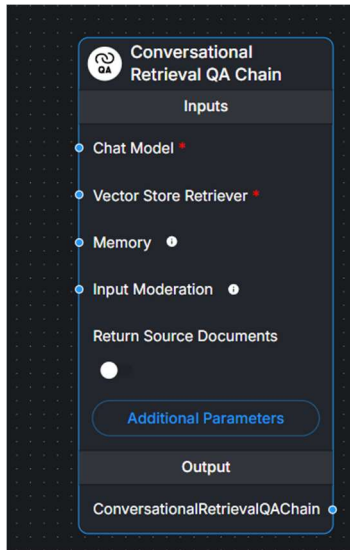
You will see an empty canvas with Untitled Chatflow



4. Build a ChatBot application

Select Add Nodes -> LangChain -> Chains -> **Conversational Retrieval QA Chain**, then drag into the canvas.

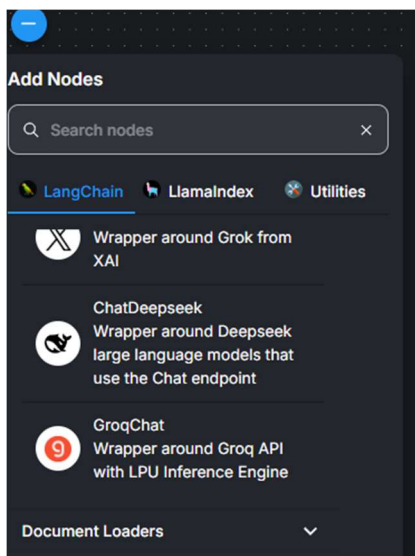




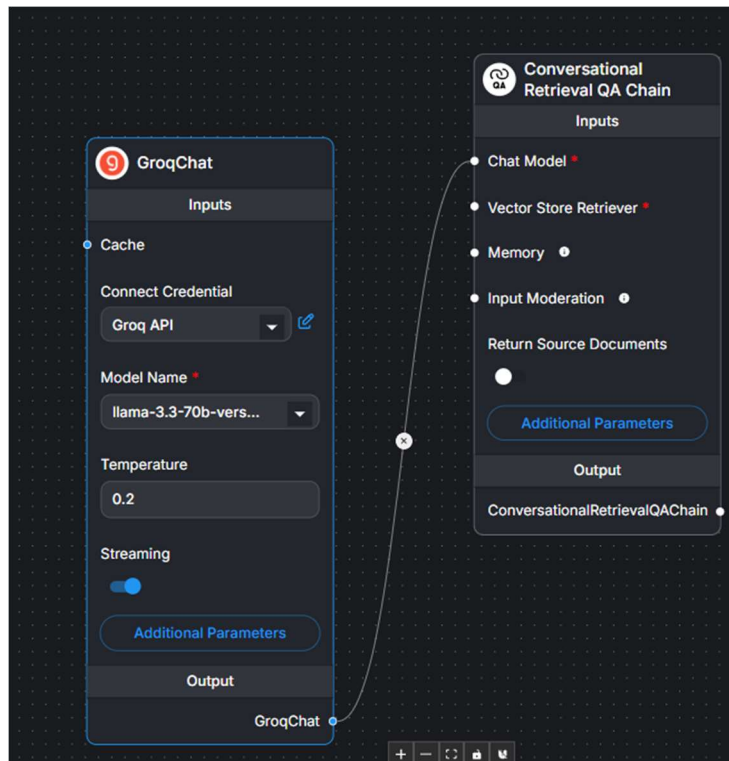
Any items with red dot are mandatory.

5. Add LLM Chat Model to Conversation Chain

Select Add Nodes->LangChain->Chat Model-> **GroqChat**, drag to the canvas. Connect it to the Conversation Chain



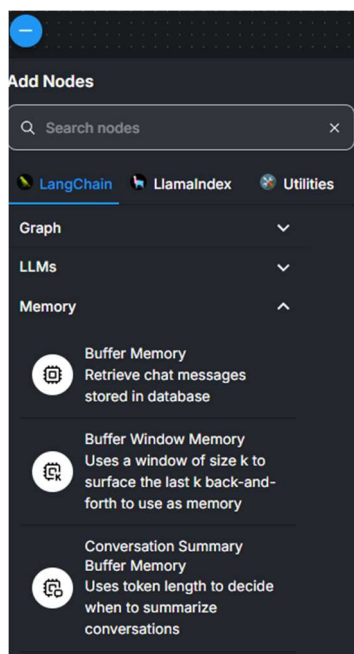
Connect Credential: Groq API

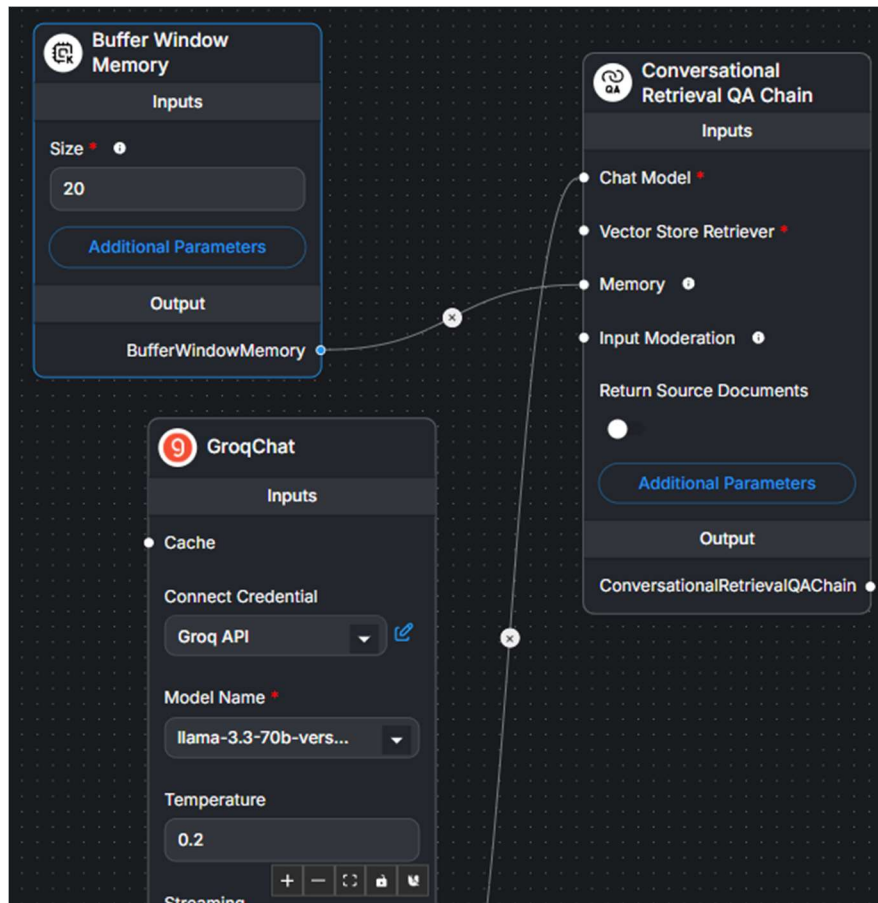


Mode Name: llama 3.3 70b
Verstile
Temperature : 0.2

6. Add LLM Memory

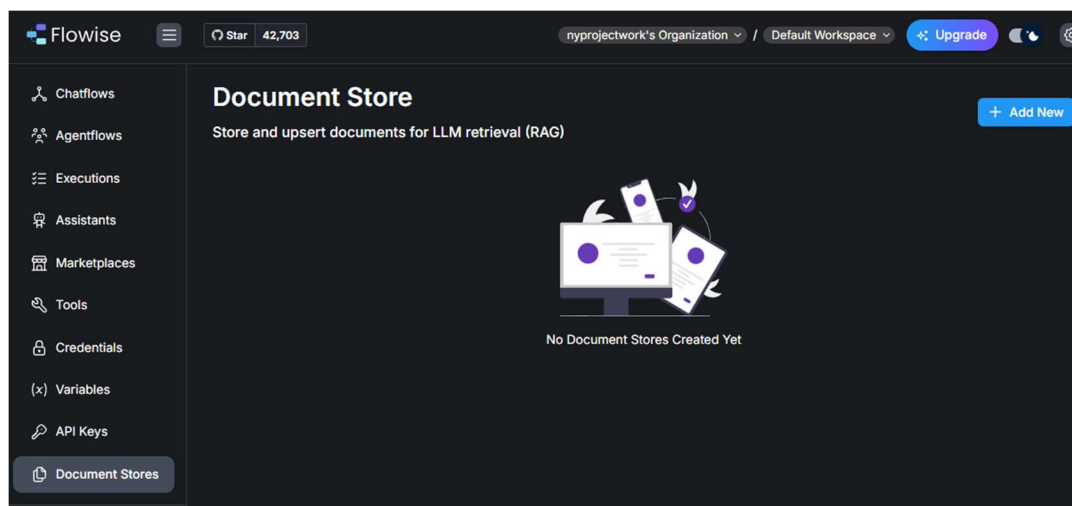
Select Add Nodes->LangChain->Memory-> **Buffer Memory Window**, drag to the canvas. Connect it to the Conversation Chain. Set Buffer Window Memory Size to 20.





7. Add data into Document Store


Select main menu->Document Stores



Select Document Stores-> Add New

Document Store

Store and upsert documents for LLM retrieval (RAG)

 Add New Document Store

Name *



Retro Bites Diner

Description

Cancel Add

Document Store

Store and upsert documents for LLM retrieval (RAG)

  [+ Add New](#)

Retro Bites Diner ○ EMPTY


📄 0 flow 📄 0 chars 📄 0 chunks

Items per page: 12 Items 1 to 1 of 1 < 1 >

← Retro Bites Diner

[+ Add Document Loader](#) [More Actions ▾](#)

○ EMPTY

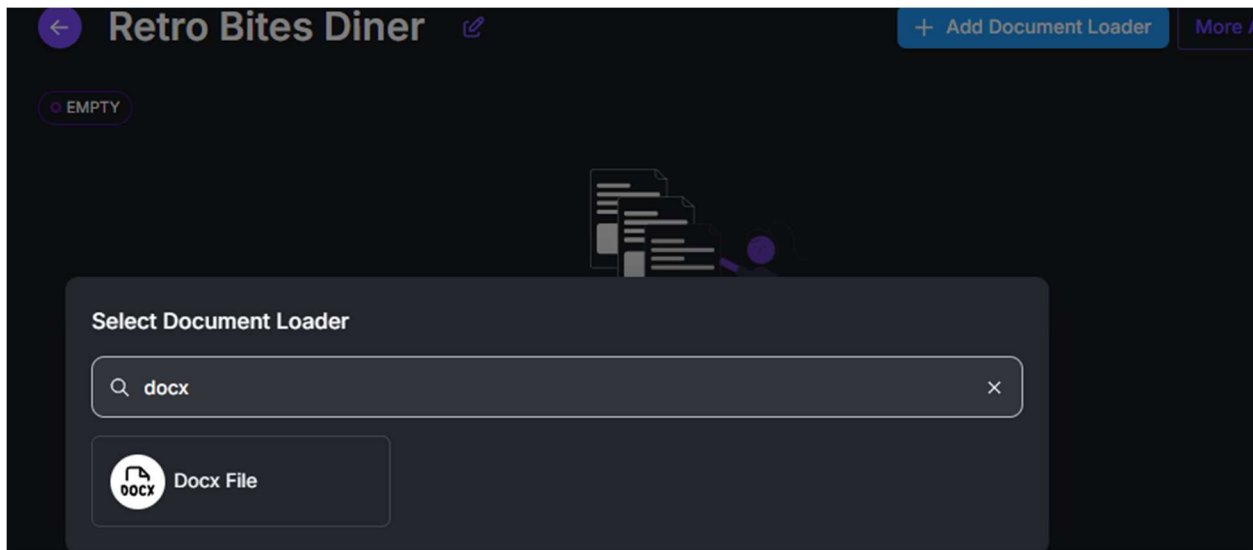


No Document Added Yet

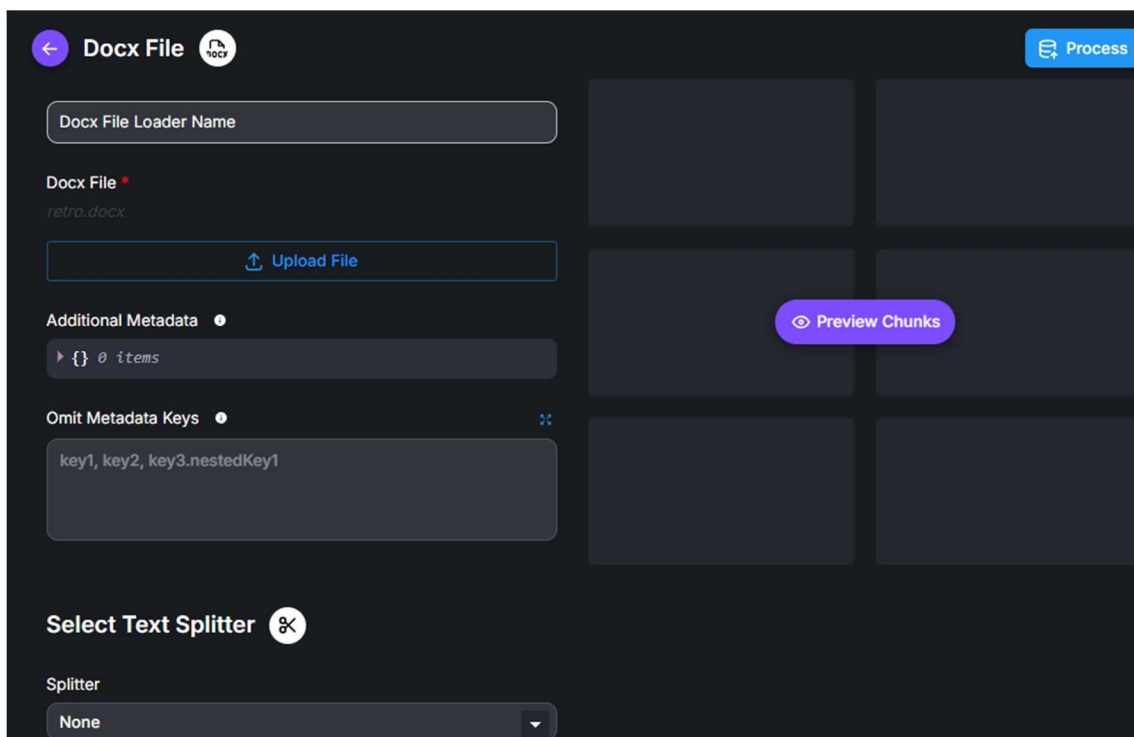
[+ Add Document Loader](#)

Select Add Document Loader.

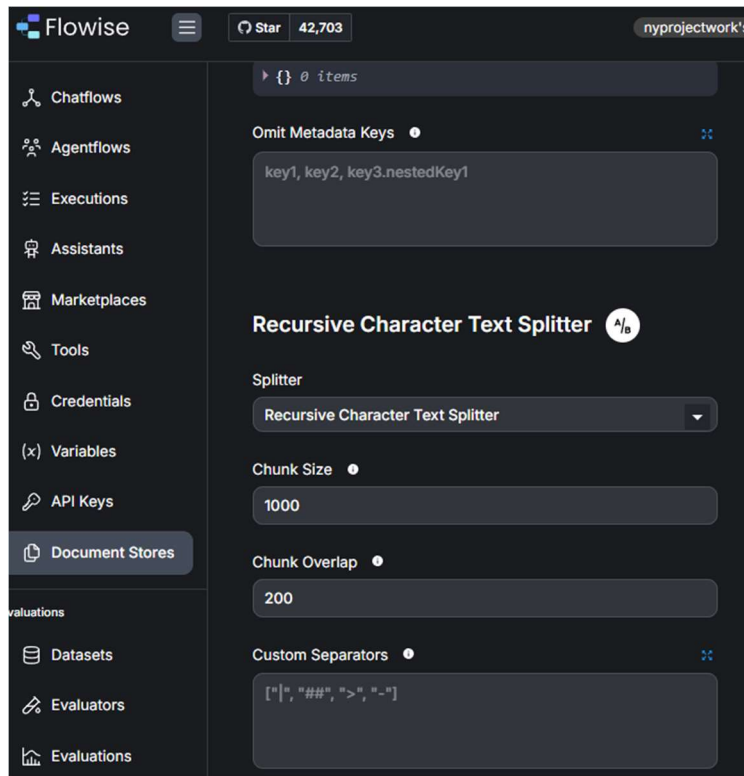
At Select Document Loader- Search for Docx File.



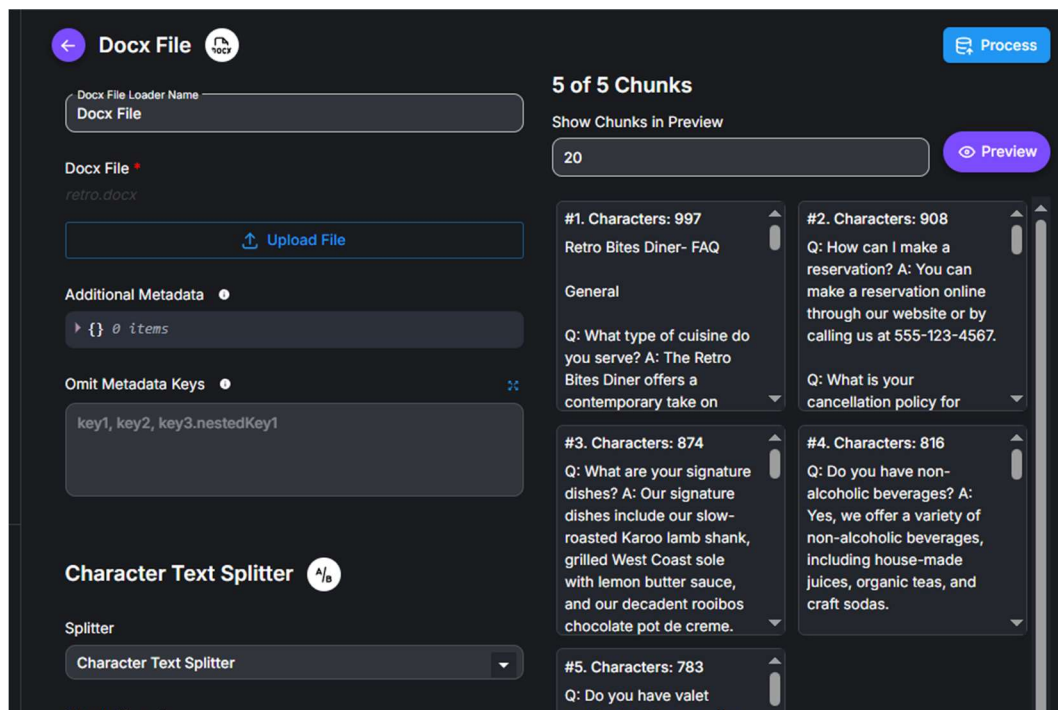
Upload file **Retro.docx**



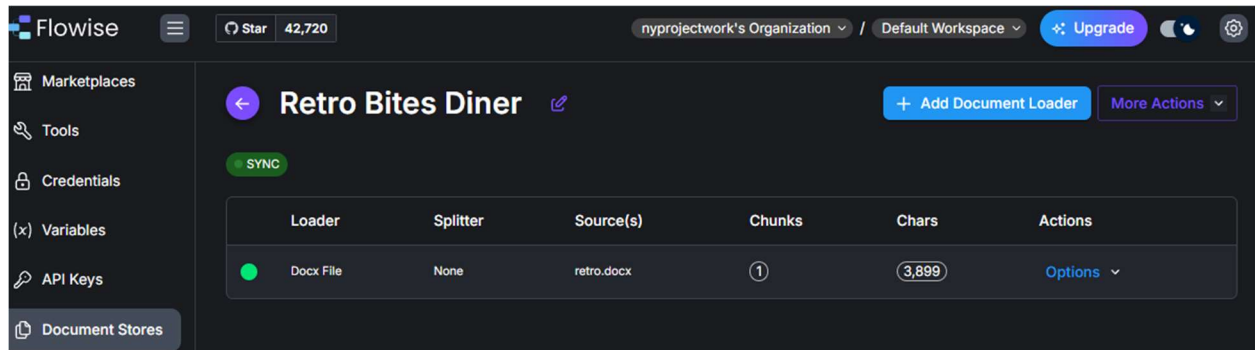
Set the Splitter-> Recursive Character Text Splitter



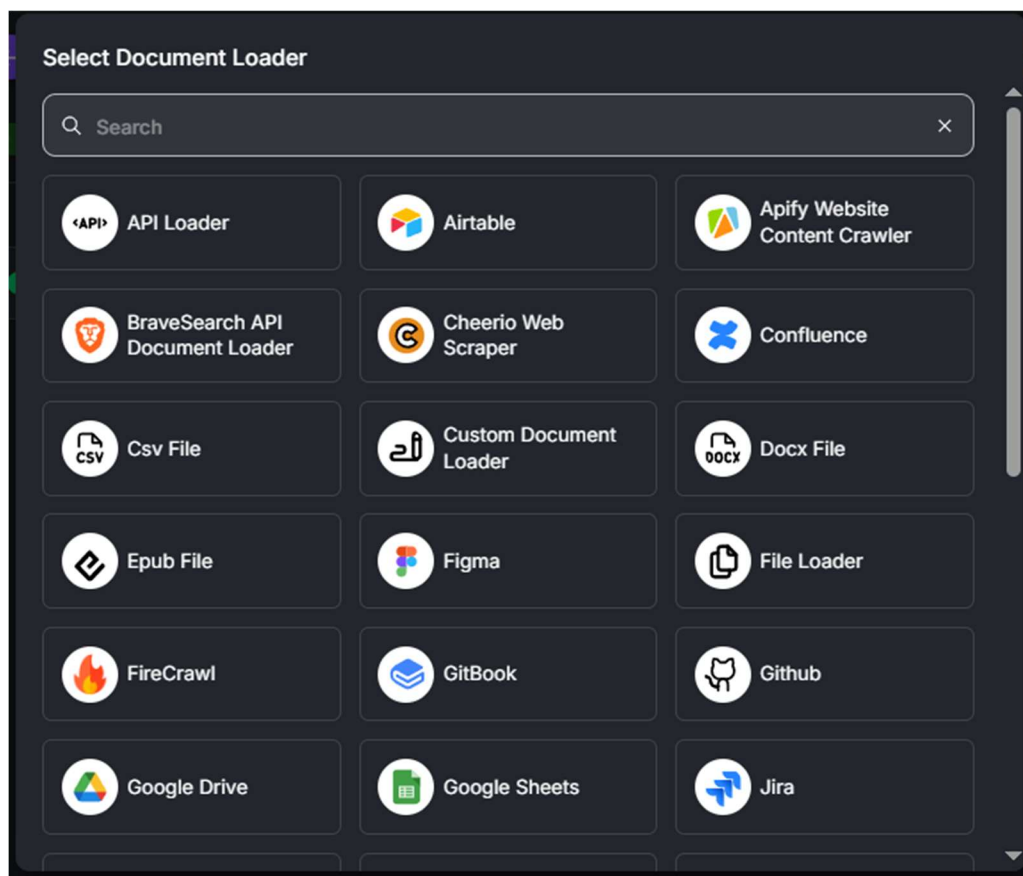
Select Process



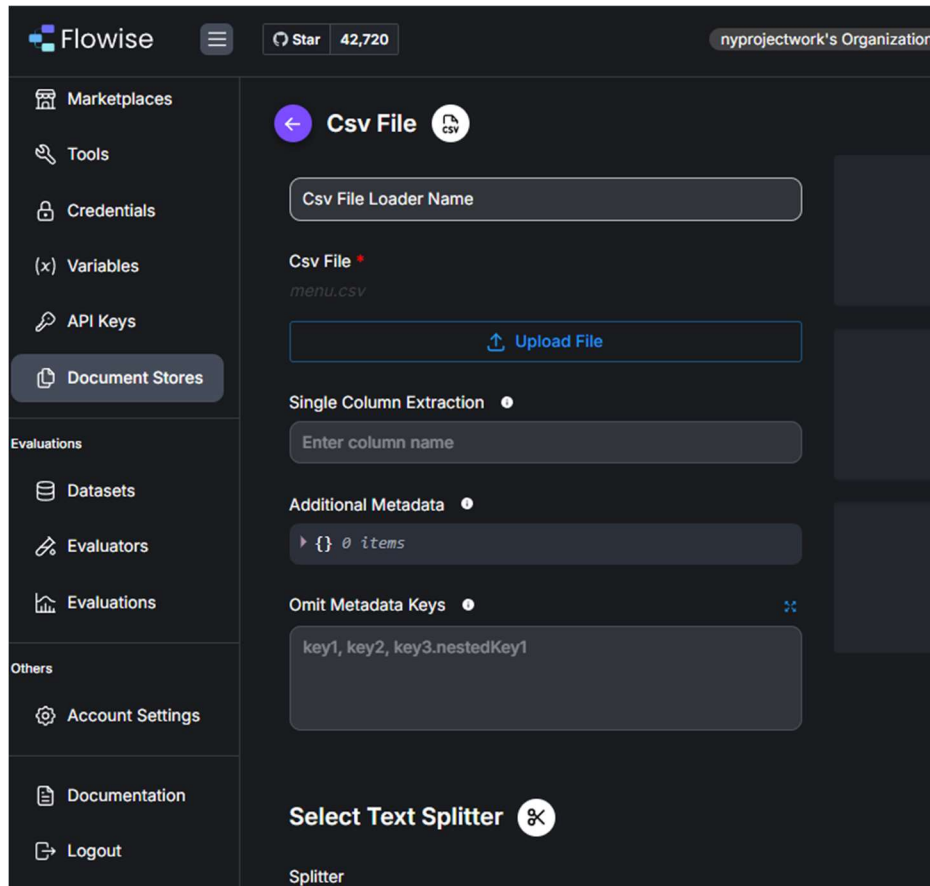
Upon completion of the document store for the docx file. Select +Add Document Loader once again. This time we will upload a csv file contain of the food prices.



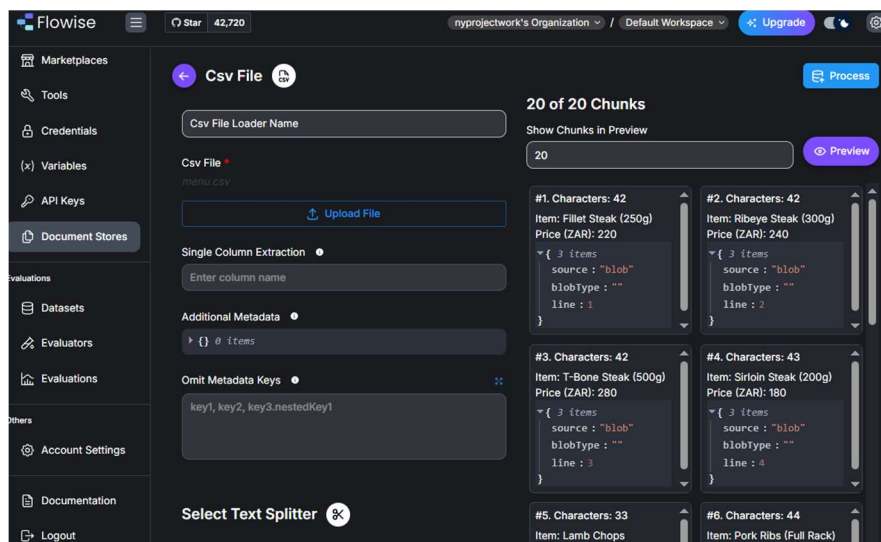
Select Csv File



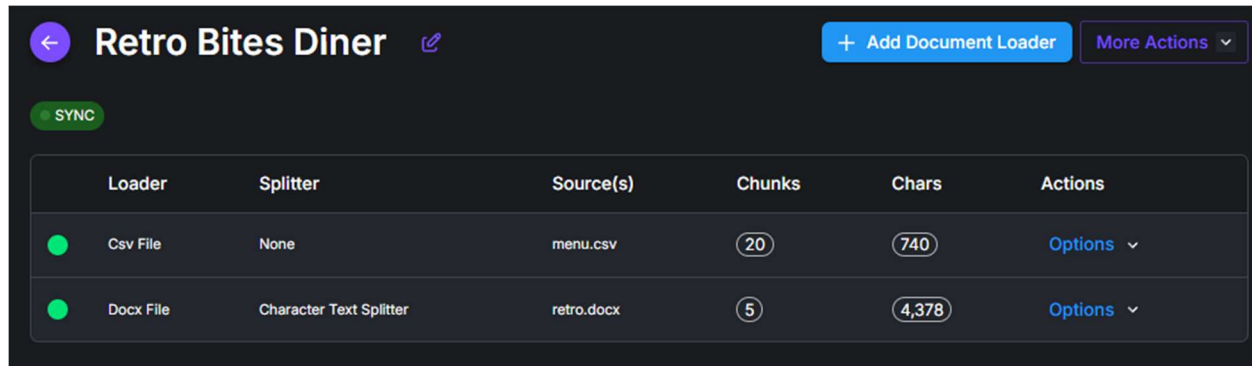
Upload the CSV file -> menu.csv. And no Text Splitter is required because the information is organized row by row.



Select Preview Chunk



Select Process to store data in document store. All processes are complete there will be green dot at each loader.



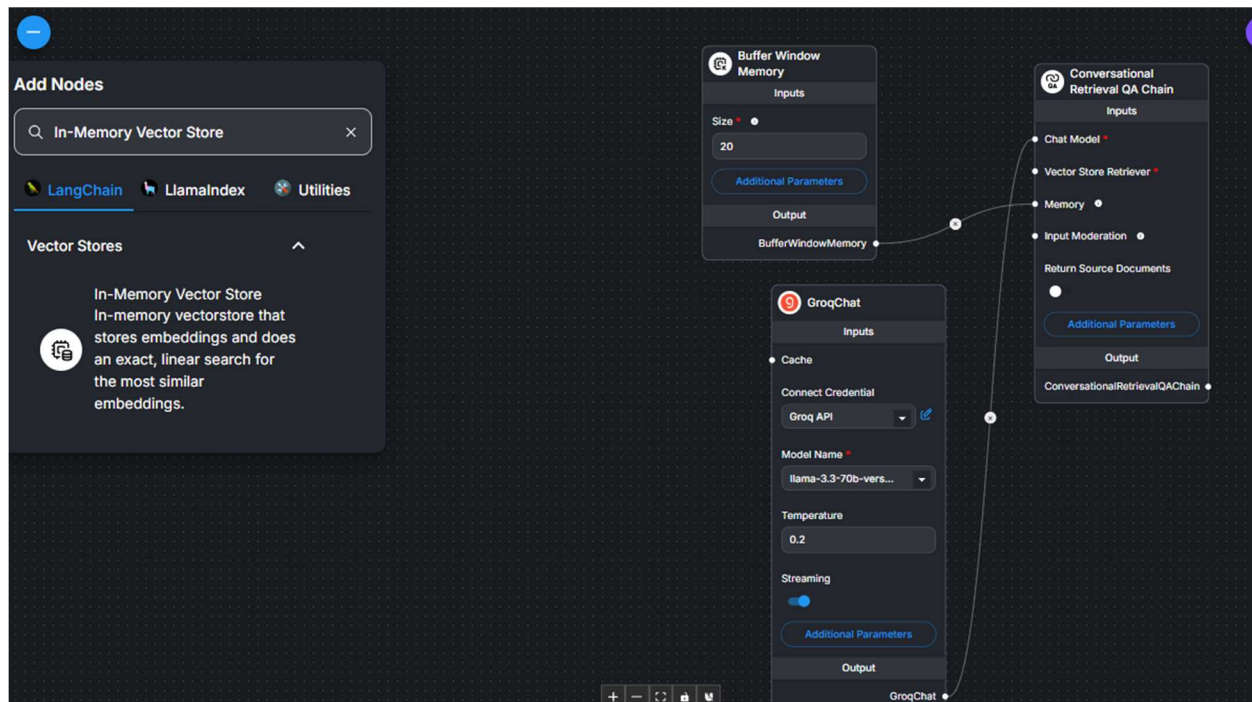
The screenshot shows the 'Retro Bites Diner' interface with a table of document loaders. The table has columns for Loader, Splitter, Source(s), Chunks, Chars, and Actions. Two loaders are listed: 'Csv File' and 'Docx File', both with green status dots.

Loader	Splitter	Source(s)	Chunks	Chars	Actions
Csv File	None	menu.csv	20	740	Options ▾
Docx File	Character Text Splitter	retro.docx	5	4,378	Options ▾

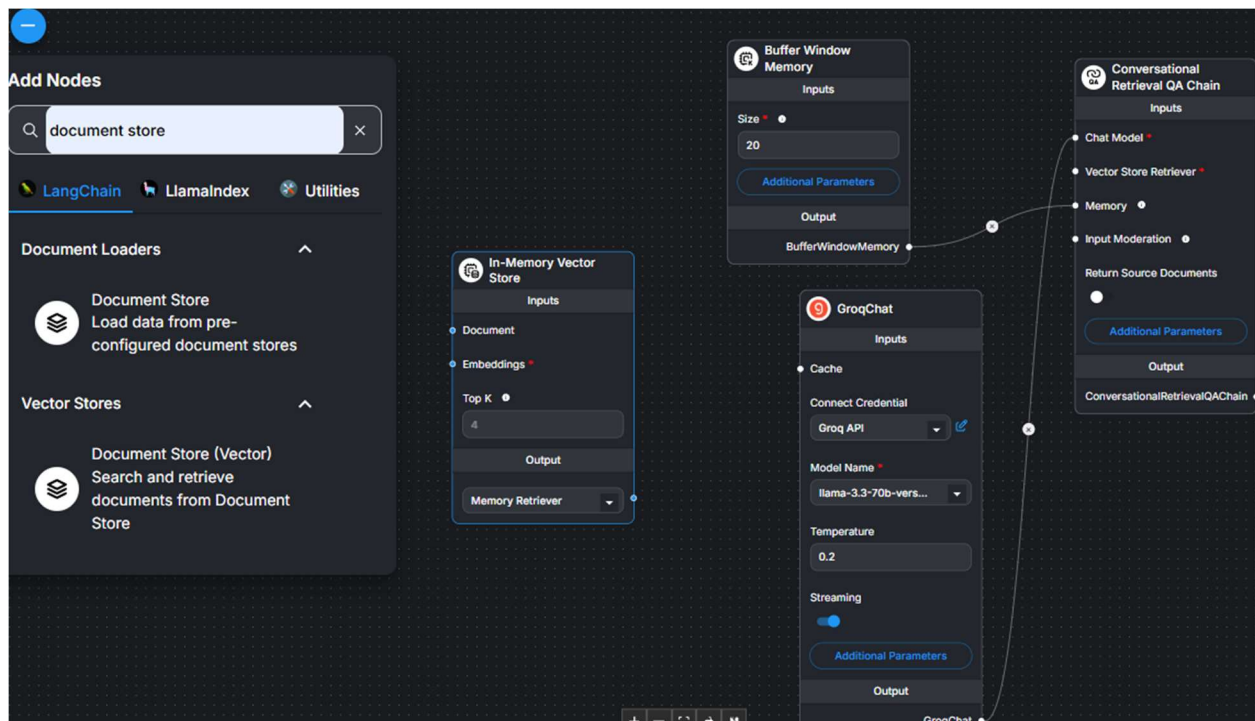
8. Setup Vector store Retrieval

The chunked data need to store as the vector store in order to used as the retrieval of information based on the customer query.

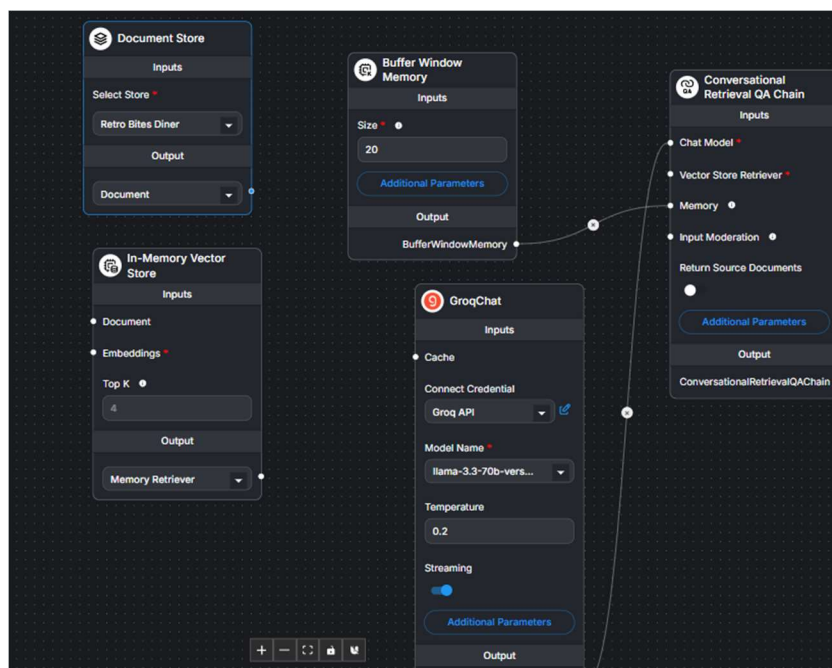
Back to Chatflow, search for **In-Memory Vector Store**, drag into canvas



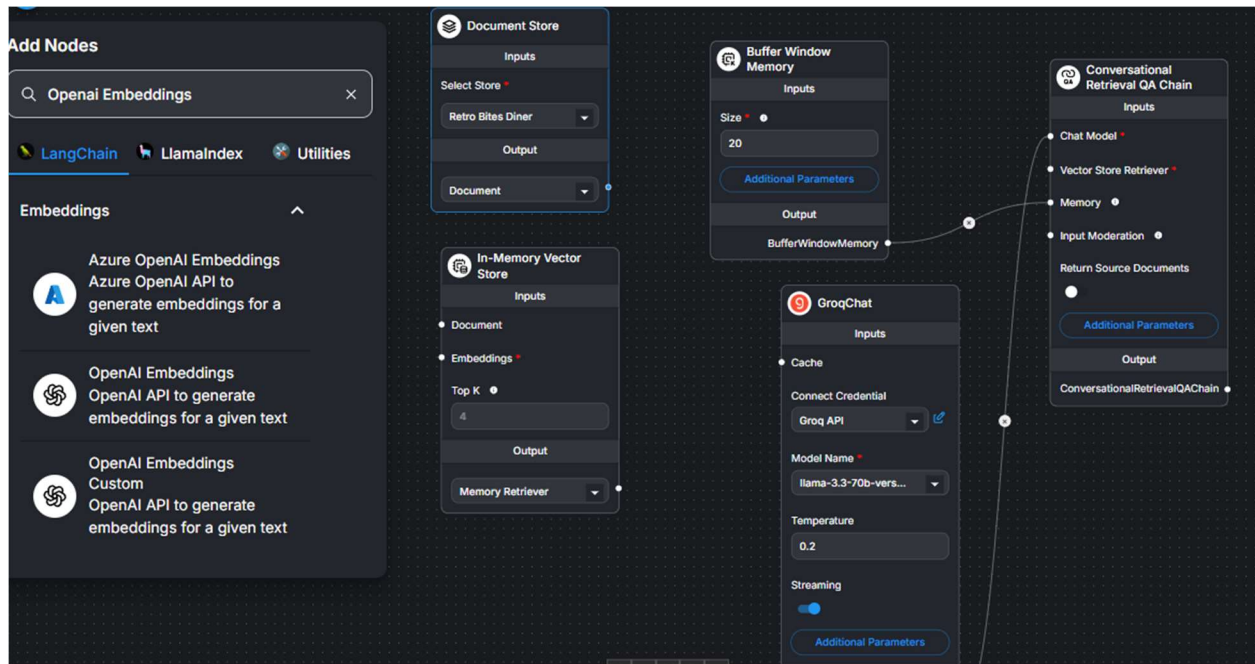
Search for document store. Select Document Loader-> **Document Store**, drag into canvas.



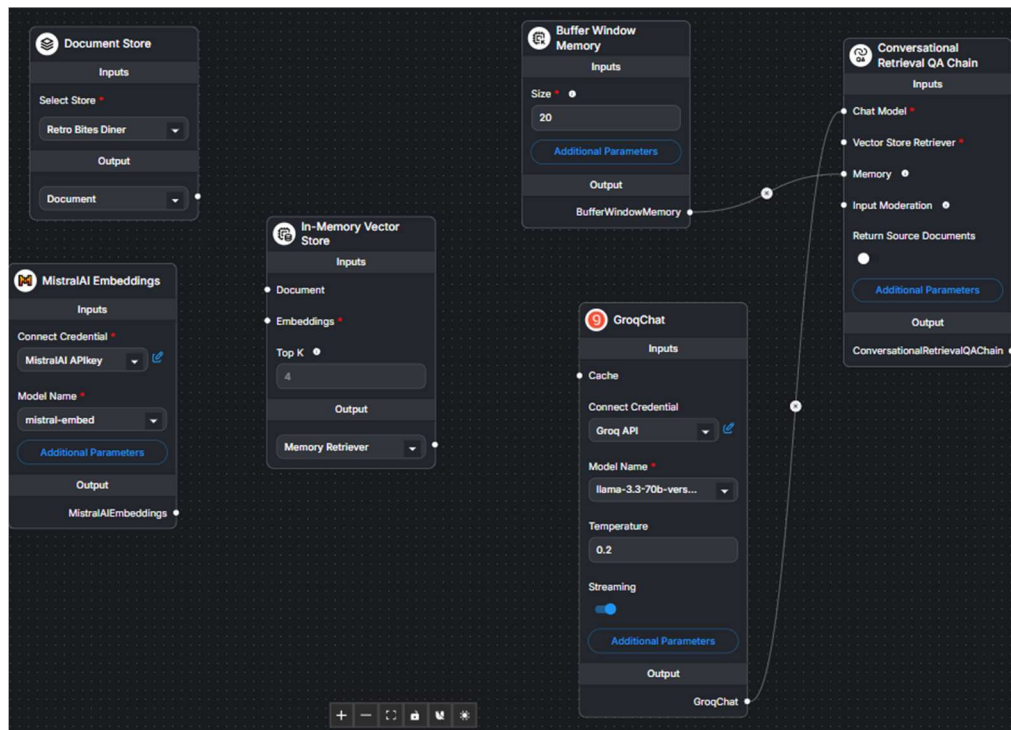
In the Document Store, Select Store-> **Retro Bites Diner** (this the document store we created previously)



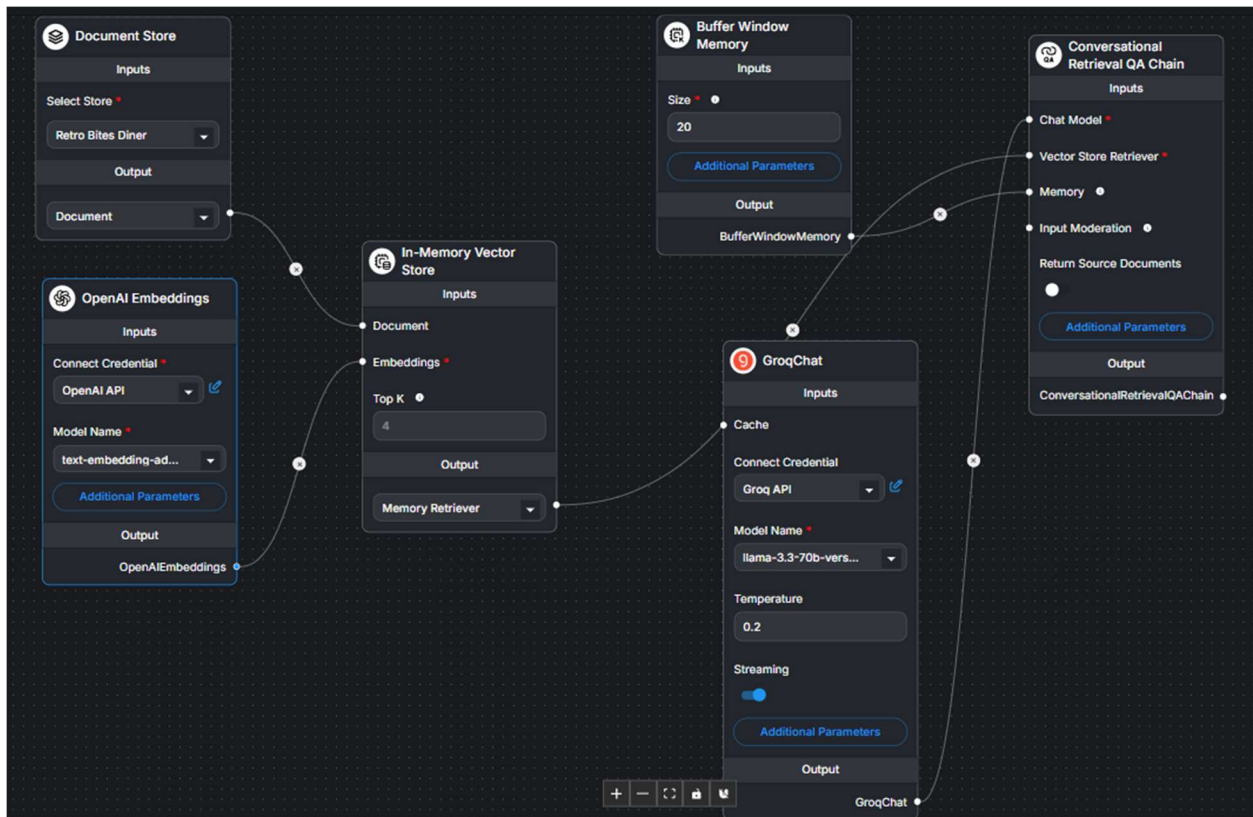
Next Search for **Openai Embeddings**. Drag into canvas



At the MistralAI Embeddings, Connection Credential -> MistralAI API. And Model Name-> mistral-embed.



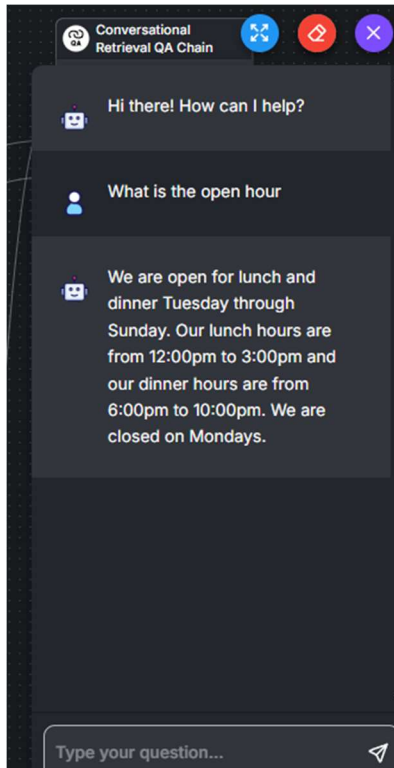
Complete all the connections as seen below



9. Test the Chatbot

Select the Chatbot button at top right.

Enter a question: What is open hour



Enter question: What is your specialty dish?

