



## **Assignment 2**

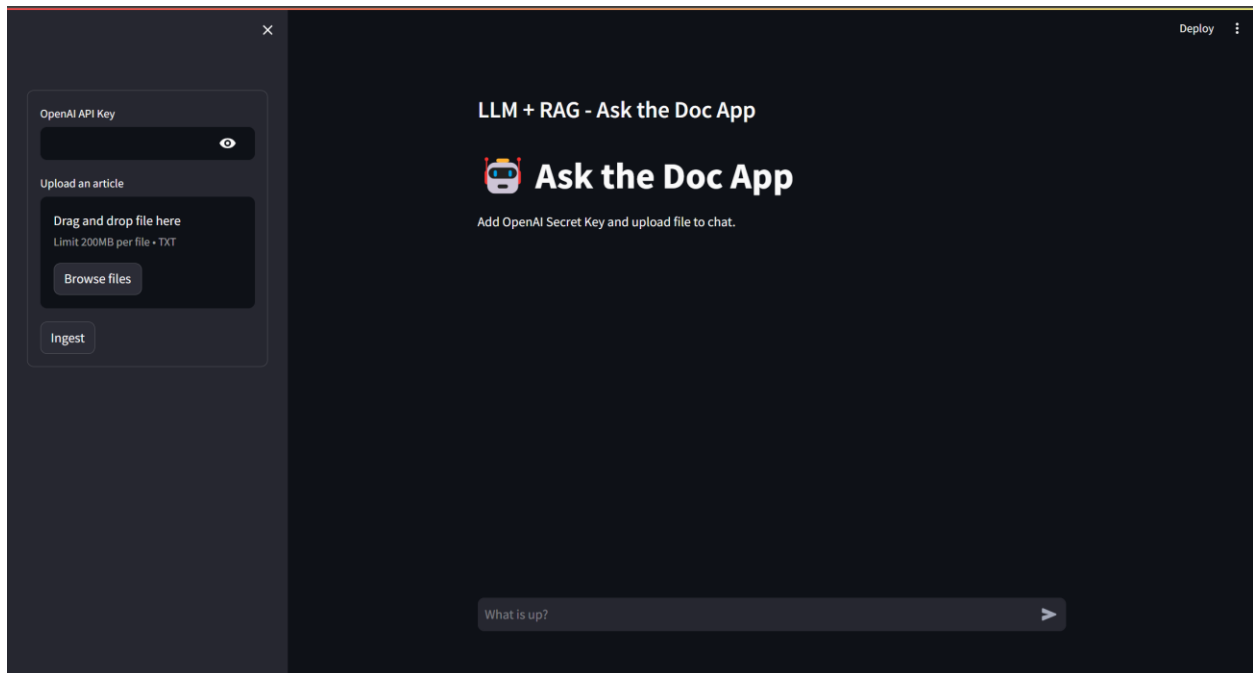
**Building a ChatGPT App with  
Retrieval Augmented Generation using Streamlit.io**

**Akshay Bharadva [100943365]**

**KNOWLEDGE AND EXP SYSTEMS – Durham College**

**Prof. Saber Amini**

## About Application:

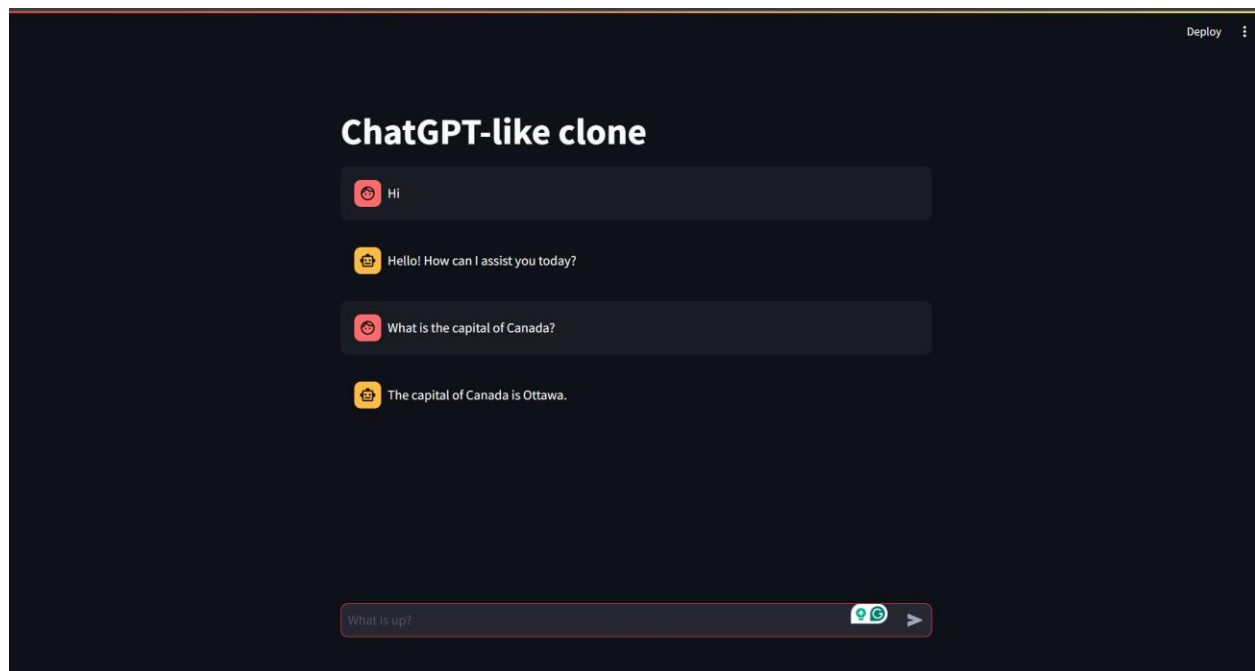


Well, **Ask the Doc App**, helps to add your OpenAI API Key and ingest information/data in txt file format. And this ingested file will be stored into Vector Database in this case/application I've used ChromaDB.

- Future enhancement
  - Add feature to upload different file formats such as .txt, .md, .pdf, .doc, and .png/.jpg/.jpeg (image format)
  - Instead of simple text-based response we can also add feature which will convert user's query into different response, like text-to-text, text-to-sql, text-to-image, or much more advanced RAG techniques.

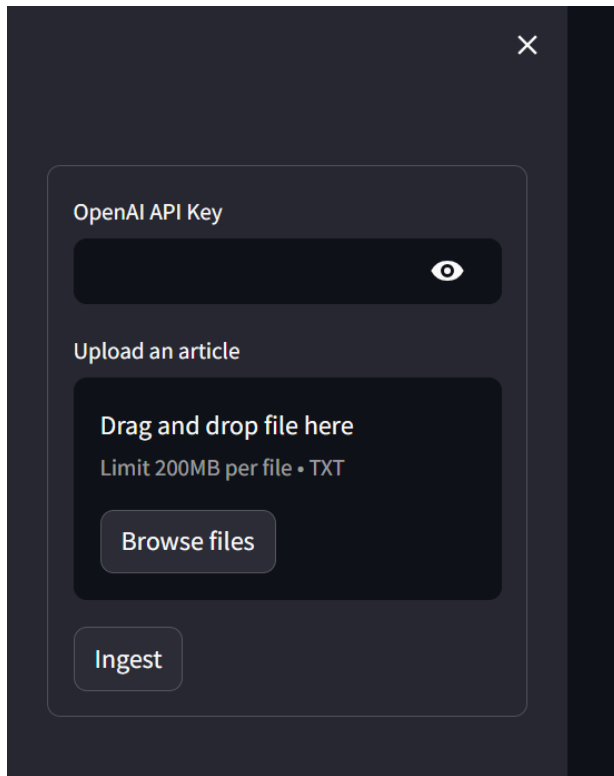
List of operations to develop **Stream.io** Application:

1. Installed libraries such as streamlit, langchain, openai, chromadb, and tiktoken
2. Imported libraries
3. Followed Streamlit.io's tutorial article - **Build a basic LLM chat app**
  - <https://docs.streamlit.io/knowledge-base/tutorials/build-conversational-apps>



4. Understood the concept of RAG and how it's used in industries – gained knowledge from multiple sources and documents/articles published by other developers and experts.
  - <https://www.databricks.com/glossary/retrieval-augmented-generation-rag>
  - <https://www.youtube.com/watch?v=tcqEUSNCn8I>
5. Take a deep look at streamlit.io's documentation and studied about other features provided by it.
  - <https://docs.streamlit.io/>

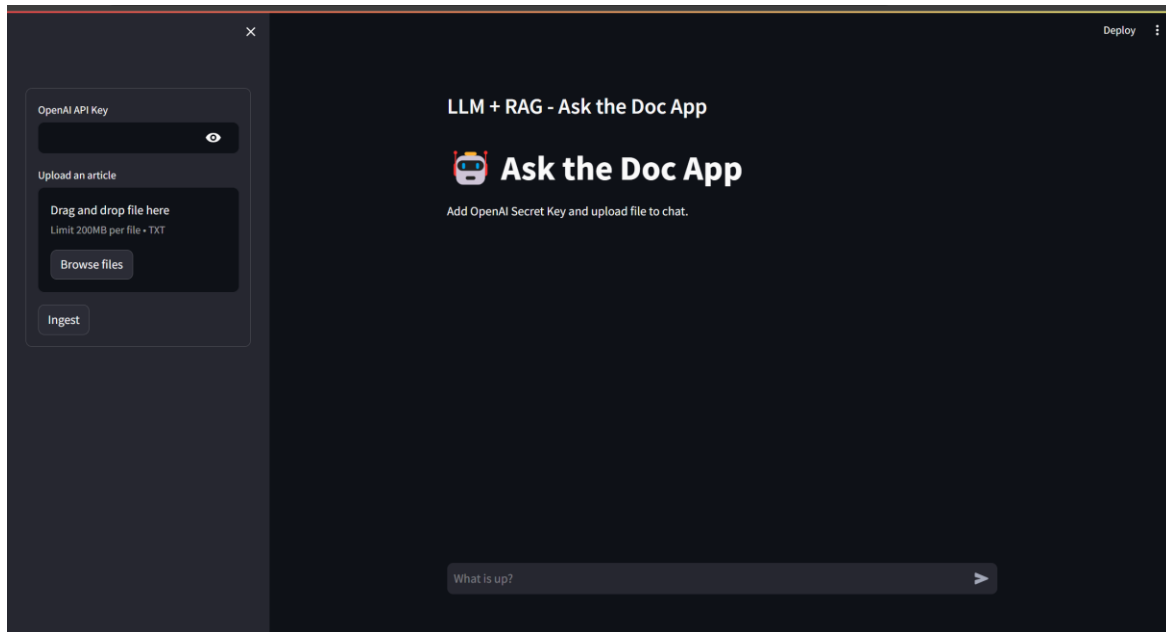
6. Developed the basic chat interface with sidebar
7. Added one password-based input field provided by streamlit so user can add their OpenAI API Key and can test/interact with the application.
8. Added the upload .txt file, streamlit file input, field so user can upload .txt file and ingest into the vector database.



The screenshot shows a dark-themed sidebar interface. At the top, there is a close button (X). Below it, the section 'OpenAI API Key' contains a password input field with an eye icon. The next section, 'Upload an article', features a large dark box with the text 'Drag and drop file here' and 'Limit 200MB per file • TXT'. A 'Browse files' button is located within this box. At the bottom of the sidebar is an 'Ingest' button.

```
st.set_page_config(page_title='🤖 Ask the Doc App')
st.subheader('LLM + RAG - Ask the Doc App')
with st.sidebar:
    with st.form('myform', clear_on_submit=False):
        openai_api_key = st.text_input('OpenAI API Key', type='password')
        uploaded_file = st.file_uploader('Upload an article', type='txt')
        submitted = st.form_submit_button('Ingest')
        if openai_api_key and openai_api_key.startswith('sk-') and uploaded_file:
            os.environ['OPENAI_API_KEY'] = openai_api_key
            with st.spinner('Ingesting the file ...'):
                result = ingest_document(uploaded_file)
            if result:
                st.success('File ingested successfully!')
```

9. Similar to ChatGPT ui added Chat interface into streamlit application. In that chat interface user can query on the text information/data ingested from the sidebar.



```
st.title('🤖 Ask the Doc App')

# chat
✓ if not retriever:
    ...st.write(f'Add OpenAI Secret Key and upload file to chat.')

    client = OpenAI(api_key=os.environ['OPENAI_API_KEY'])

✓ if "openai_model" not in st.session_state:
    ...st.session_state["openai_model"] = "gpt-3.5-turbo"

✓ if "messages" not in st.session_state:
    ...st.session_state.messages = []

✓ for message in st.session_state.messages:
    ...with st.chat_message(message["role"]):
    ...    st.markdown(message["content"])

✓ if prompt := st.chat_input("What is up?", disabled=not retriever):
    ...st.session_state.messages.append({"role": "user", "content": prompt})
    ...with st.chat_message("user"):
    ...    st.markdown(prompt)

    ...with st.chat_message("assistant"):
    ...    response = generate_response(prompt)
    ...    st.write(response)
    ...st.session_state.messages.append({"role": "assistant", "content": response})
```

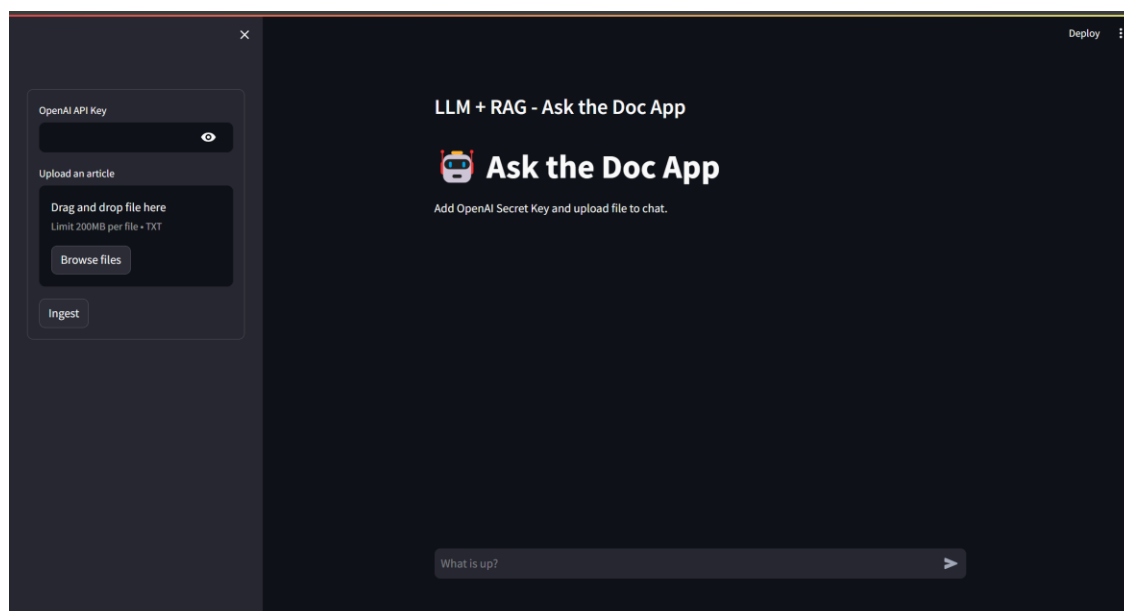
10. Created/developed two functions one to ingest uploaded file into the vector Chroma database.

```
def ingest_document(uploaded_file):  
    ....# Load document if file is uploaded  
    ....if uploaded_file is not None:  
    ....    documents = [uploaded_file.read().decode()]  
    ....# Split documents into chunks  
    ....text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)  
    ....texts = text_splitter.create_documents(documents)  
    ....# Select embeddings  
    ....embeddings = OpenAIEmbeddings(openai_api_key=os.environ['OPENAI_API_KEY'])  
    ....# Create a vectorstore from documents  
    ....db = Chroma.from_documents(texts, embeddings)  
    ....# Create retriever interface  
    ....global retriever  
    ....retriever = db.as_retriever()  
    ....return retriever
```

11. Another one to generate response based on the prompt provided by the user.

```
def generate_response(query_text):  
    ....# Create QA chain  
    ....openai_api_key=os.environ['OPENAI_API_KEY']  
    ....print(openai_api_key)  
    ....qa = RetrievalQA.from_chain_type(llm=OpenAI(openai_api_key=openai_api_key), chain_type='stuff', retriever=retriever, )  
    ....return qa.run(query_text)
```

12. Here is the final output of developed application.



## References:

- <https://www.databricks.com/glossary/retrieval-augmented-generation-rag>
- <https://docs.streamlit.io/knowledge-base/tutorials/build-conversational-apps>
- <https://www.youtube.com/watch?v=tcqEUSNCn8I>
- <https://medium.com/predict/crafting-an-ai-powered-chatbot-for-finance-using-rag-langchain-and-streamlit-4384a8076960>
- <https://medium.com/snowflake/building-a-rag-based-blog-ai-assistant-using-streamlit-openai-and-llamaindex-304f6ffe6757>