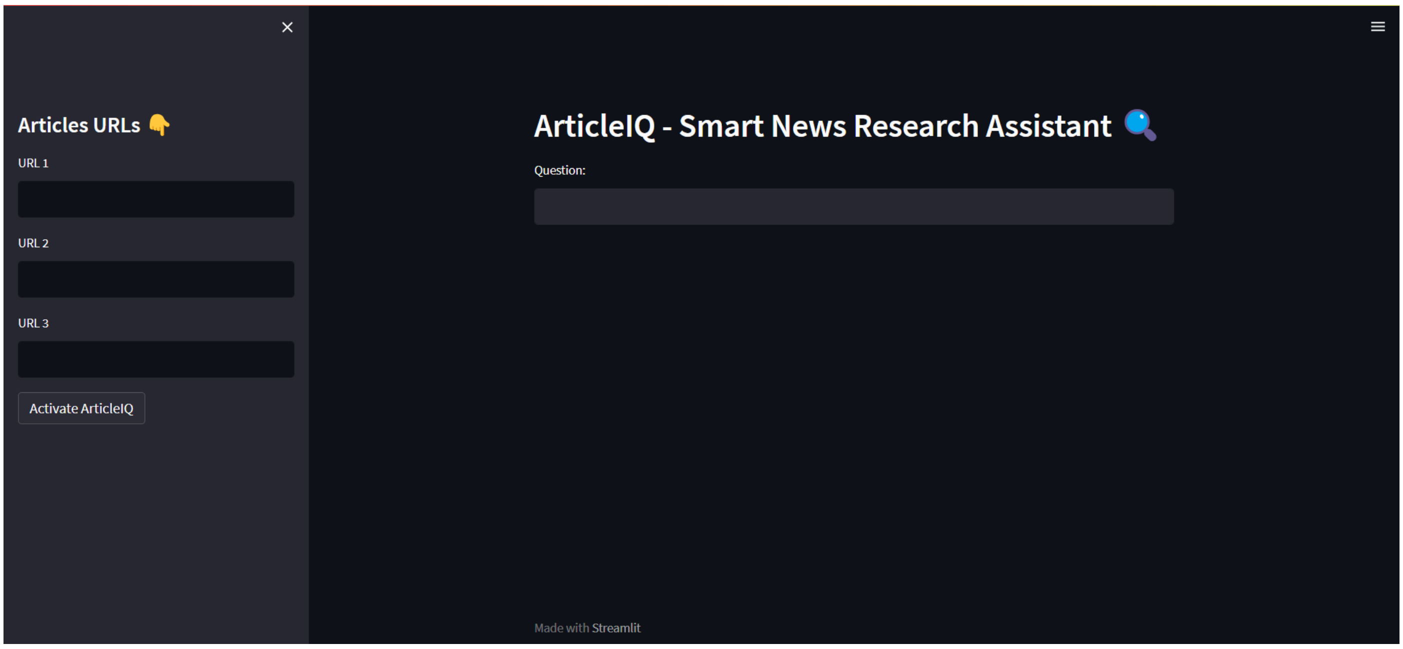
🔎 ArticleIQ Project Documentation 🔗

Title: ArticleIQ - Smart News Research Assistant

Introduction

The ArticleIQ is a Smart News Research Assistant application powered by OpenAI GPT-3. It's designed to find answers to user queries based on the content of provided articles. The application is built using the Streamlit framework and LangChain library.



Installation

Before running the application, ensure that all required libraries are installed. These can be installed using pip:

```
pip install -r requirements.txt
```

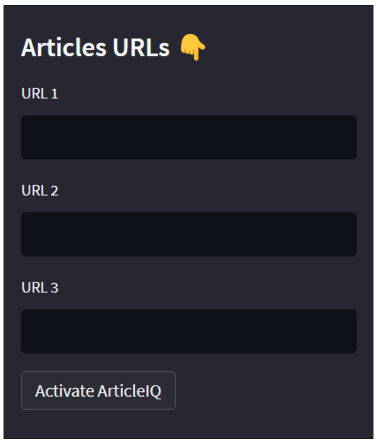
File Structure

The main script is main.py, which contains all the code for the application.

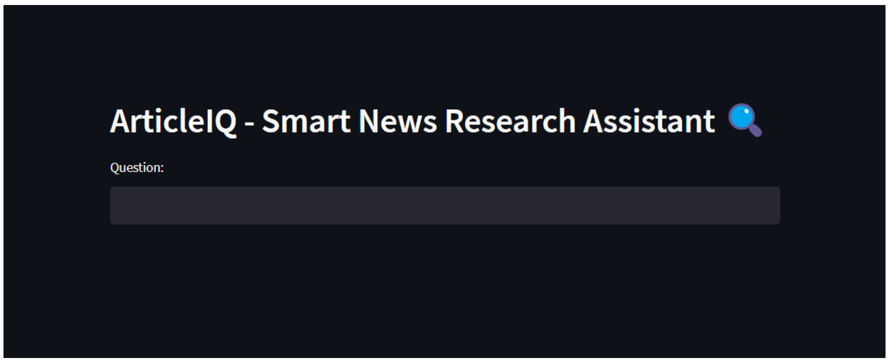
```
streamlit run main.py
```

How to Use

When the application is run, a Streamlit interface is displayed in the web browser. The user can enter up to 3 URLs of online articles in the sidebar. After entering the URLs, the user can click the "Activate ArticleIQ" button to start the data processing.



Once the data is processed, the user can type their question in the 'Question:' input field. The application will then find the most relevant answer from the content of the provided articles.



Detailed Code Explanation

The script starts by importing the necessary modules and initializing the Streamlit interface.

```python
# Import necessary libraries
import os
import streamlit as st
import pickle
import time
from langchain import OpenAI
from langchain.document_loaders import UnstructuredURLLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
from langchain.chains import RetrievalQAWithSourcesChain
from dotenv import load_dotenv
```

The user is then prompted to enter up to 3 articles URLs. Once the "Activate ArticleIQ" button is clicked, the following steps are taken:

The URLs are loaded using the **UnstructuredURLLoader** class from the Langchain library.

```
# Load data from URLs
loader = UnstructuredURLLoader(urls = urls)
status_display.text('Loading Data ⏳')
data = loader.load()
```

The loaded data is split into chunks using the **RecursiveCharacterTextSplitter** class. This helps in handling large amounts of text.

```
# Split data into manageable chunks
status_display.text('Splitting Data ✂️')
text_splitter = RecursiveCharacterTextSplitter( separators= ['\n\n', '\n', '.', ','], chunk_size = 1000)
individual_chunks = text_splitter.split_documents(data)
```

Each chunk is then embedded using the **OpenAIEmbeddings** class and saved into a FAISS index using the FAISS class.

```
# Embed chunks and save to FAISS index
embeddings = OpenAIEmbeddings()
vector_data = FAISS.from_documents(individual_chunks, embeddings)
status_display.text('Embedding Vectors 📩 📩')
time.sleep(2)
```

The FAISS index is then saved to a pickle file **'FAISS_Vector_Data.pkl'** for later use.

```
# Save FAISS index to pickle file
with open(file_path, "wb") as fp:
    pickle.dump(vector_data, fp)
```

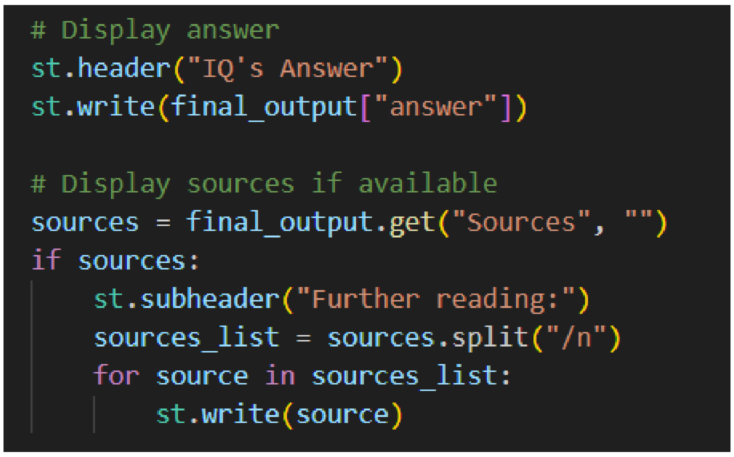
After the user enters a question, the application checks if the pickle file exists.

```
# Collect question from user
question = status_display.text_input('Question: ')
# If a question is entered
if question:
    # If FAISS index pickle file exists
    if os.path.exists(file_path):
        # Load FAISS index from pickle file
        with open(file_path, 'rb') as fp:
            vector_store = pickle.load(fp)
```

If it does, it loads the FAISS index from the file and creates a retrieval chain using the **RetrievalQAWithSourcesChain** class. This chain is used to find the most relevant answer to the user's question from the article content.

```
# Initialize retrieval chain
retrieval_chain = RetrievalQAWithSourcesChain.from_llm(llm = llm, retriever = vector_store.as_retriever())
# Find answer to question
final_output = retrieval_chain({"question": question}, return_only_outputs = True)
```

The answer is then displayed on the interface along with the sources from where the information was retrieved.



```
# Display answer
st.header("IQ's Answer")
st.write(final_output["answer"])

# Display sources if available
sources = final_output.get("Sources", "")
if sources:
    st.subheader("Further reading:")
    sources_list = sources.split("/n")
    for source in sources_list:
        st.write(source)
```

Conclusion

The ArticleIQ project is a powerful tool for performing research based on online articles. It leverages the power of OpenAI GPT-3 to find precise answers to user queries and also will summarize the context as per the question. Despite its simplicity, it can be a very useful tool for journalists, researchers etc...