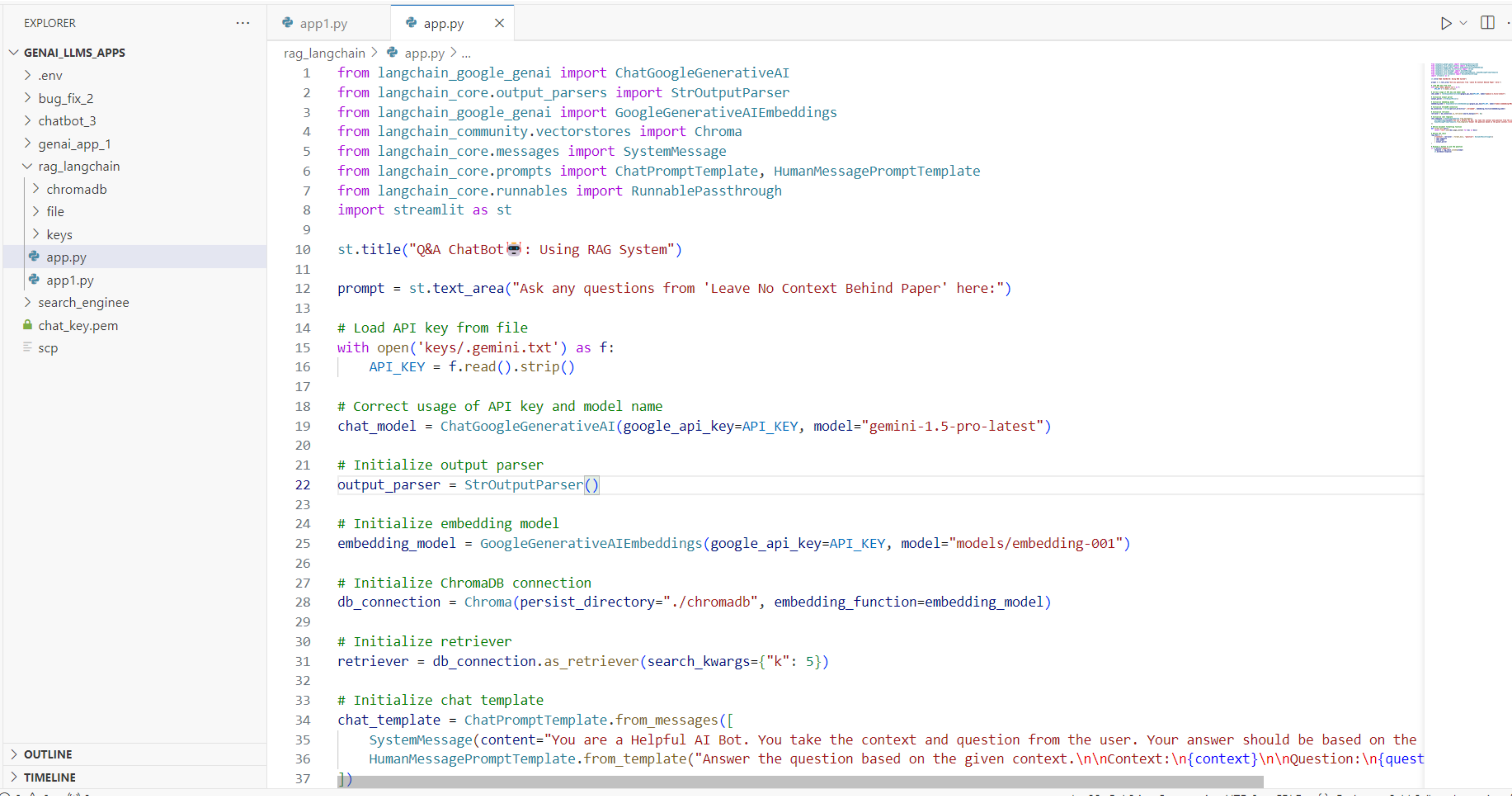
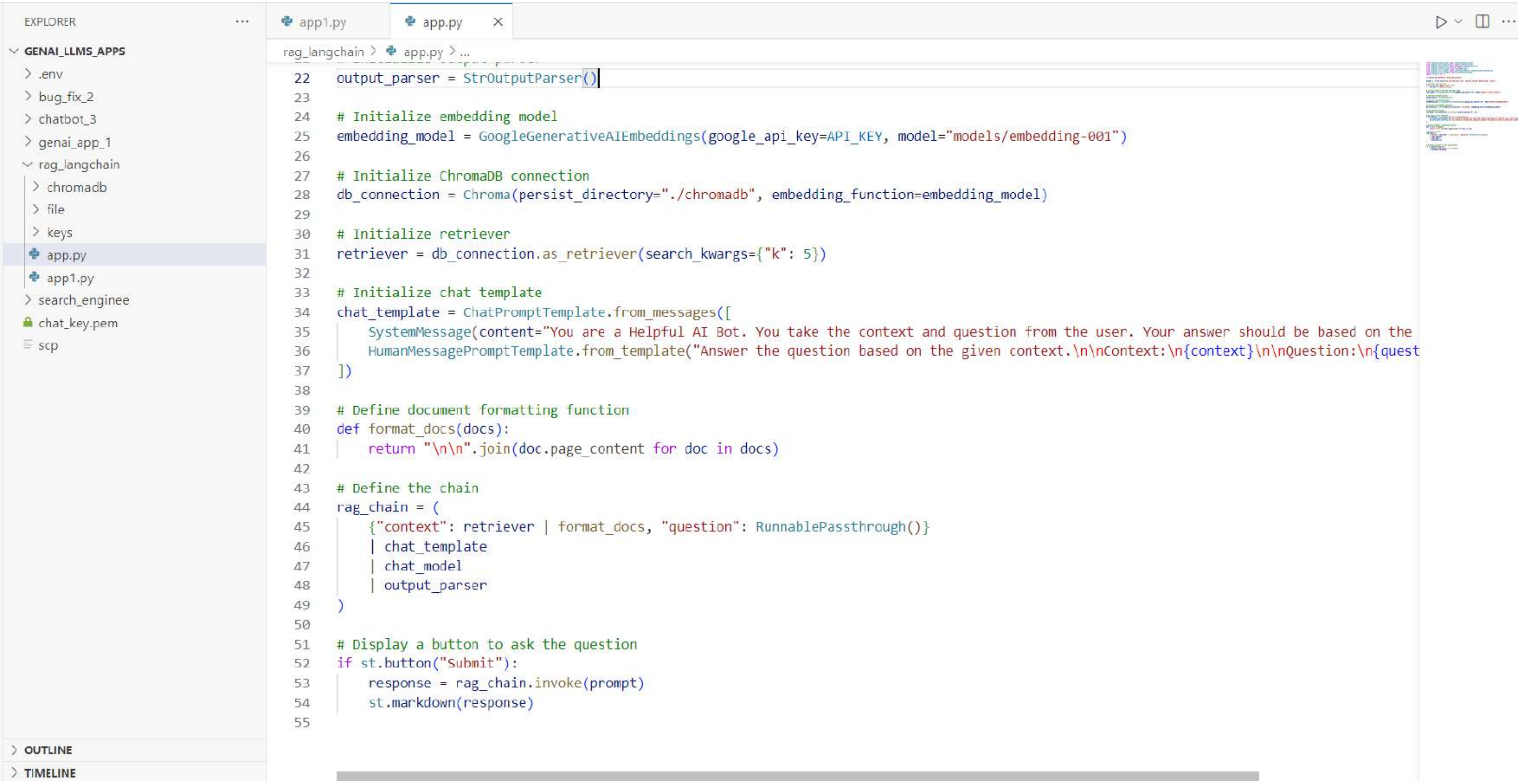
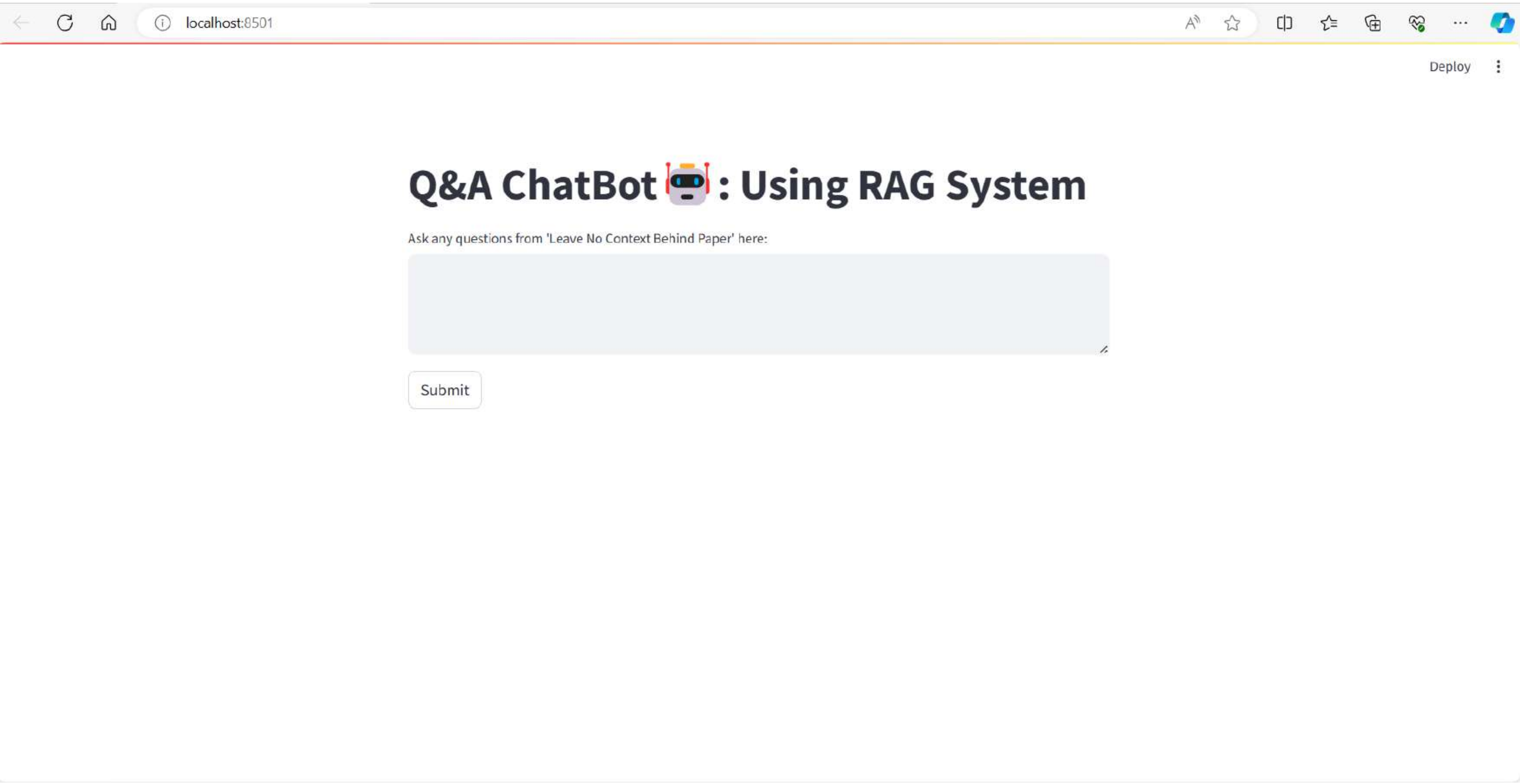


```python
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.output_parsers import StrOutputParser
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain_community.vectorstores import Chroma
from langchain_core.messages import SystemMessage
from langchain_core.prompts import ChatPromptTemplate, HumanMessagePromptTemplate
from langchain_core.runnables import RunnablePassthrough
import streamlit as st

st.title("Q&A ChatBot🤖: Using RAG System")

prompt = st.text_area("Ask any questions from 'Leave No Context Behind Paper' here:")

# Load API key from file
with open('keys/.gemini.txt') as f:
    API_KEY = f.read().strip()

# Correct usage of API key and model name
chat_model = ChatGoogleGenerativeAI(google_api_key=API_KEY, model="gemini-1.5-pro-latest")

# Initialize output parser
output_parser = StrOutputParser()

# Initialize embedding model
embedding_model = GoogleGenerativeAIEmbeddings(google_api_key=API_KEY, model="models/embedding-001")

# Initialize ChromaDB connection
db_connection = Chroma(persist_directory="./chromadb", embedding_function=embedding_model)

# Initialize retriever
retriever = db_connection.as_retriever(search_kwargs={"k": 5})

# Initialize chat template
chat_template = ChatPromptTemplate.from_messages([
    SystemMessage(content="You are a Helpful AI Bot. You take the context and question from the user. Your answer should be based on the
    HumanMessagePromptTemplate.from_template("Answer the question based on the given context.\n\nContext:\n{context}\n\nQuestion:\n{quest
])
```



```python
output_parser = StrOutputParser()

# Initialize embedding model
embedding_model = GoogleGenerativeAIEmbeddings(google_api_key=API_KEY, model="models/embedding-001")

# Initialize ChromaDB connection
db_connection = Chroma(persist_directory="./chromadb", embedding_function=embedding_model)

# Initialize retriever
retriever = db_connection.as_retriever(search_kwargs={"k": 5})

# Initialize chat template
chat_template = ChatPromptTemplate.from_messages([
    SystemMessage(content="You are a Helpful AI Bot. You take the context and question from the user. Your answer should be based on the
    HumanMessagePromptTemplate.from_template("Answer the question based on the given context.\n\nContext:\n{context}\n\nQuestion:\n{quest
])

# Define document formatting function
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

# Define the chain
rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | chat_template
    | chat_model
    | output_parser
)

# Display a button to ask the question
if st.button("Submit"):
    response = rag_chain.invoke(prompt)
    st.markdown(response)
```

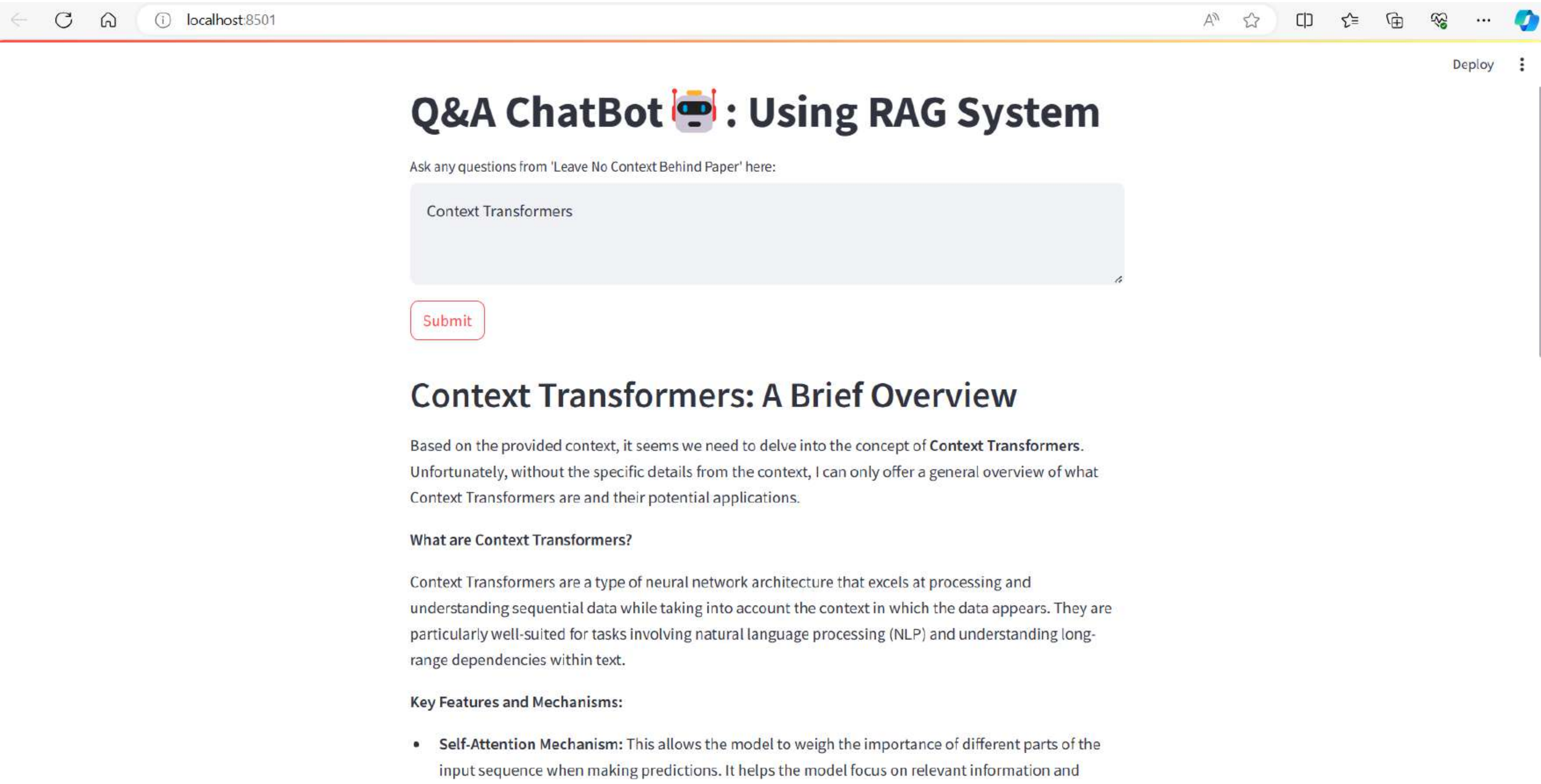

localhost:8501

Deploy

Q&A ChatBot 🤖: Using RAG System

Ask any questions from 'Leave No Context Behind Paper' here:

Submit



Q&A ChatBot 🤖 : Using RAG System

Ask any questions from 'Leave No Context Behind Paper' here:

Context Transformers

Context Transformers: A Brief Overview

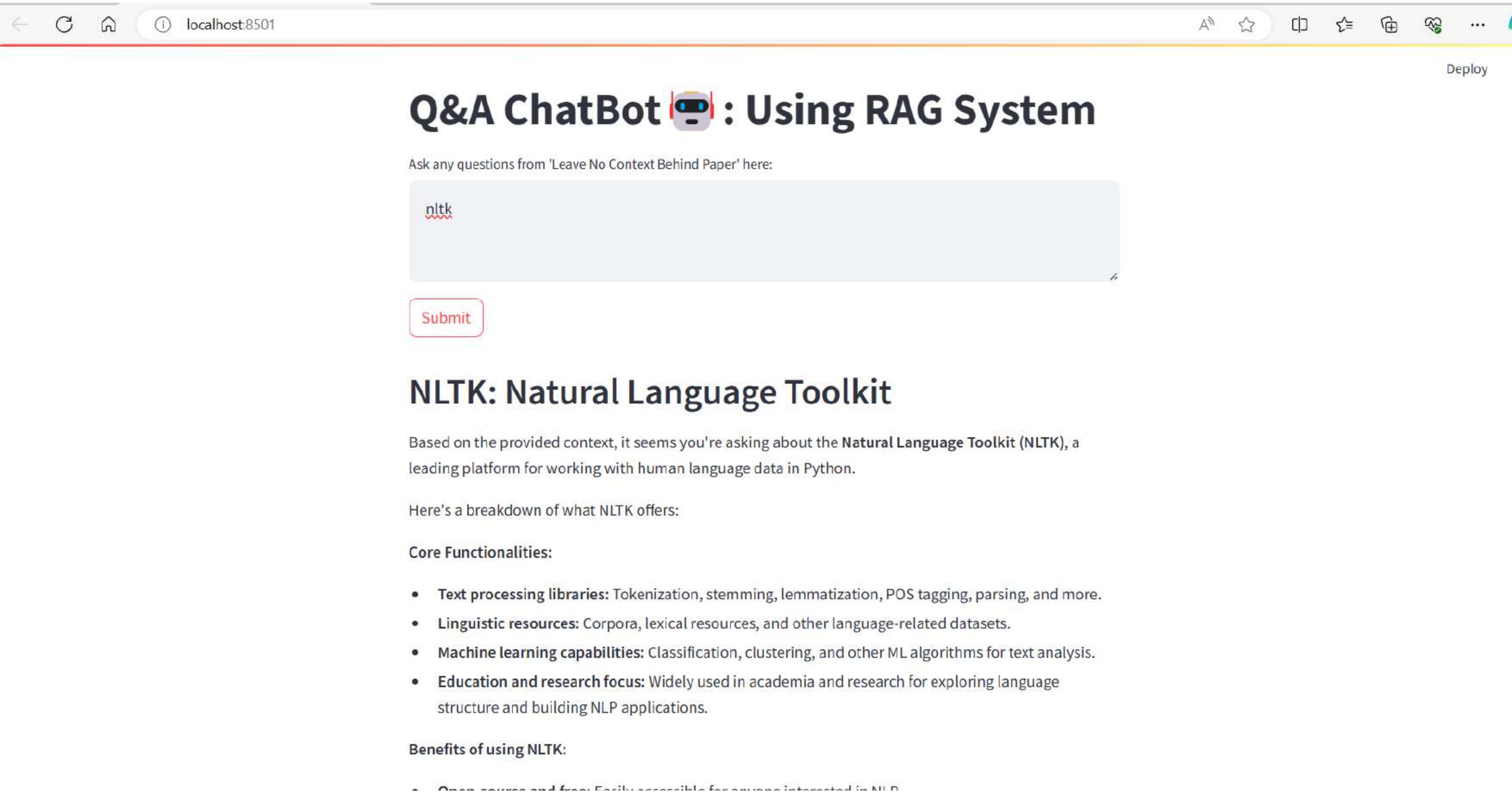
Based on the provided context, it seems we need to delve into the concept of **Context Transformers**. Unfortunately, without the specific details from the context, I can only offer a general overview of what Context Transformers are and their potential applications.

What are Context Transformers?

Context Transformers are a type of neural network architecture that excels at processing and understanding sequential data while taking into account the context in which the data appears. They are particularly well-suited for tasks involving natural language processing (NLP) and understanding long-range dependencies within text.

Key Features and Mechanisms:

- **Self-Attention Mechanism:** This allows the model to weigh the importance of different parts of the input sequence when making predictions. It helps the model focus on relevant information and



Q&A ChatBot 🤖 : Using RAG System

Ask any questions from 'Leave No Context Behind Paper' here:

nltk

NLTK: Natural Language Toolkit

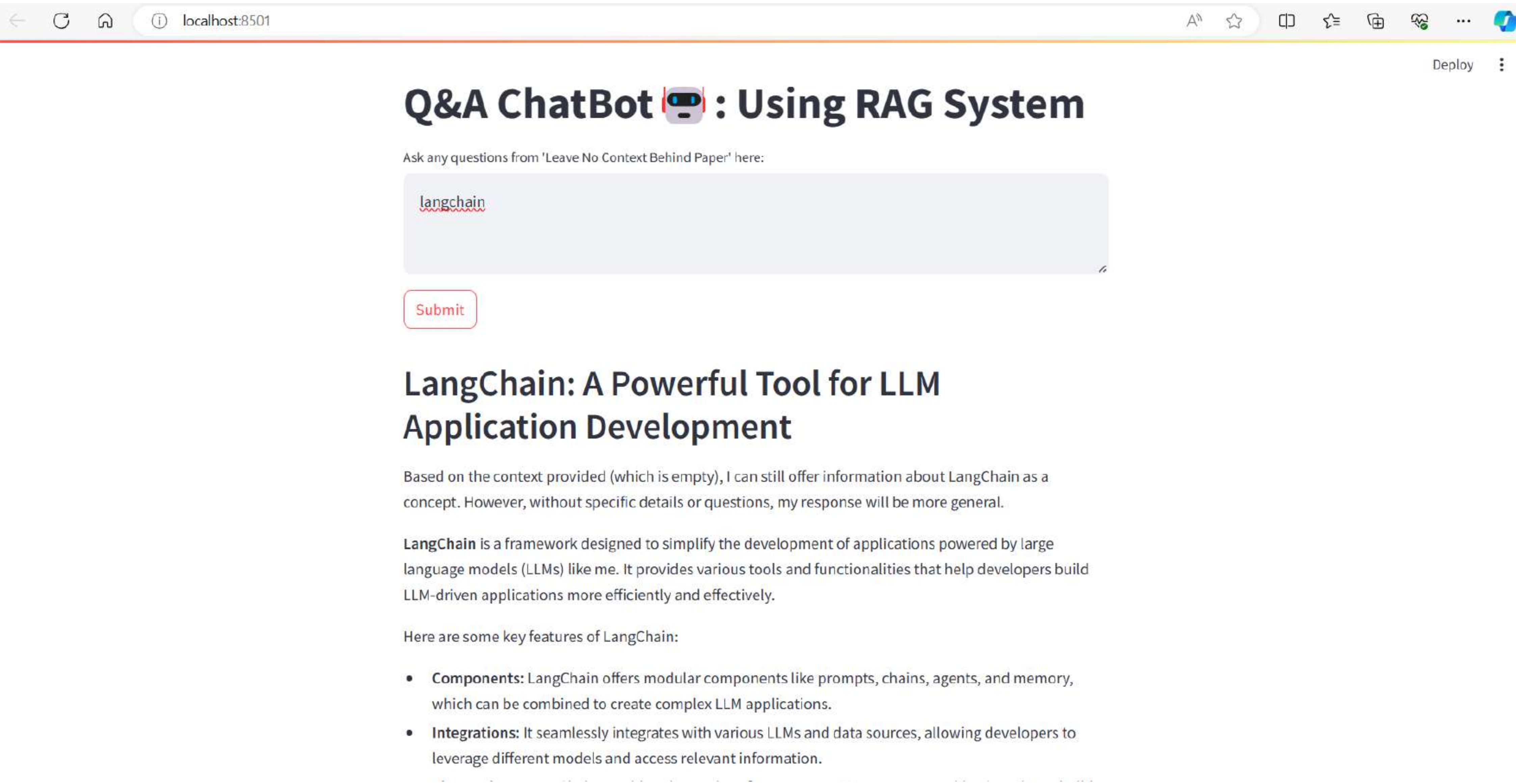
Based on the provided context, it seems you're asking about the **Natural Language Toolkit (NLTK)**, a leading platform for working with human language data in Python.

Here's a breakdown of what NLTK offers:

Core Functionalities:

- **Text processing libraries:** Tokenization, stemming, lemmatization, POS tagging, parsing, and more.
- **Linguistic resources:** Corpora, lexical resources, and other language-related datasets.
- **Machine learning capabilities:** Classification, clustering, and other ML algorithms for text analysis.
- **Education and research focus:** Widely used in academia and research for exploring language structure and building NLP applications.

Benefits of using NLTK:


localhost:8501

Deploy

Q&A ChatBot : Using RAG System

Ask any questions from 'Leave No Context Behind Paper' here:

langchain

Submit

LangChain: A Powerful Tool for LLM Application Development

Based on the context provided (which is empty), I can still offer information about LangChain as a concept. However, without specific details or questions, my response will be more general.

LangChain is a framework designed to simplify the development of applications powered by large language models (LLMs) like me. It provides various tools and functionalities that help developers build LLM-driven applications more efficiently and effectively.

Here are some key features of LangChain:

- **Components:** LangChain offers modular components like prompts, chains, agents, and memory, which can be combined to create complex LLM applications.
- **Integrations:** It seamlessly integrates with various LLMs and data sources, allowing developers to leverage different models and access relevant information.