Labs Data & AI Innovation Day

Lab 5: Build a Python chatbot

This document describes how to build a Python chatbot to query the Azure Cosmos DB for MongoDB vCore in natural language

# Pre-requisites

Ensure that you have the following software installed on your system before proceeding with the lab:

* Visual Studio Code: A cross-platform code editor that supports Python development. You can download it from <https://code.visualstudio.com/>
* Python 3.10.11: The latest version of the Python programming language. You can download it from <https://www.python.org/downloads/release/python-31011/>

Note: If you are using a different version of Python, make sure that it is compatible with the libraries and packages used in this lab.

* Azure OpenAI account registered in the Azure subscription used for this lab

# Create a new Python virtual environment

Follow these steps to create the environment required to build the Python chatbot

* Create a folder on your local machine e.g. C:\lab5
* Open Visual Studio Code
* Create a new file called requirements.txt
* Open requirements.txt and copy/paste the following code:

python-dotenv

tenacity==8.2.3

ipykernel

matplotlib

plotly

scikit-learn

pymongo==3.11.3

dnspython

openai==1.6.1

azure-cosmos

streamlit

langchain==0.0.352

pymongo

tiktoken

langchain\_openai==0.0.2

unstructured

* Open a new terminal: Menu Terminal > New terminal
* Create a new environment by entering python -m venv .venv
* Activate this new environment with .venv\scripts\activate
* Install library dependencies with pip install -r requirements.txt (this can take several minutes)

A screenshot of a computer

Description automatically generated

* Create a new file called .env
* Copy/paste the following code:

COSMOSDB\_MONGODB\_HOST=cosmos-mongo-vcore-2024.mongocluster.cosmos.azure.com

COSMOSDB\_MONGODB\_USERNAME=sa

COSMOSDB\_MONGODB\_PASSWORD=Password1234

COSMOSDB\_MONGODB\_DATABASE=database\_teamXX

COSMOSDB\_MONGODB\_PRODUCTS=products\_teamXX

COSMOSDB\_MONGODB\_CUSTOMERS=customers\_teamXX

COSMOSDB\_NOSQL\_ACCOUNT=cosmos-nosql-2024

COSMOSDB\_NOSQL\_DATABASE\_NAME=database\_teamXX

COSMOSDB\_NOSQL\_CONTAINER\_NAME=conversations

COSMOSDB\_NOSQL\_KEY="bIGjXkgPgPaRu96XZNQo7vK23JBXMTlxAW6exZfIKAuuVmERgk02AaUAhSY4FPyTBPf03FKwiSyFACDb8BsnjA=="

AZURE\_OPENAI\_API\_KEY=5c32a1b4740f47318378f97e33beca29

AZURE\_OPENAI\_ENDPOINT=https://teamXXopenai.openai.azure.com/

AZURE\_OPENAI\_EMBEDDING\_MODEL=text-embedding-ada-002

AZURE\_OPENAI\_CHAT\_MODEL=gpt-35-turbo

AZURE\_OPENAI\_API\_VERSION=2023-12-01-preview

AZURE\_SEARCH\_SERVICE=https://aisearch-openhack-2024.search.windows.net

AZURE\_SEARCH\_KEY=rs6r4q03EBlKTG9OEGL9Sn9G3FF1jRWGZL92dhPmqeAzSeDbrYGZ

AZURE\_SEARCH\_API\_VERSION=2023-11-01

* Replace XX with the number you were assigned for this lab

# Create a simple chatbot

In this exercice, we will create a chatbot that can answer simple questions

* In Visual Studio Code, create a file call app.py
* Copy/paste the import statements

import streamlit as st

import os, uuid

from urllib.parse import quote

from datetime import datetime

from langchain.chains import LLMChain

from langchain.prompts import PromptTemplate

from langchain\_openai import AzureChatOpenAI, AzureOpenAIEmbeddings

from langchain.schema import HumanMessage

from langchain.memory import ConversationBufferMemory, CosmosDBChatMessageHistory

from langchain.memory.chat\_message\_histories import StreamlitChatMessageHistory

from langchain.chains import ConversationalRetrievalChain

from langchain.callbacks.base import BaseCallbackHandler

from langchain.vectorstores import AzureCosmosDBVectorSearch

from dotenv import load\_dotenv

Let’s make sure that we are able to read the content of the .env file

* Copy this code to app.py

def init\_env():

    load\_dotenv()

    st.set\_page\_config(page\_title="CosmicWorks Chatbot", page\_icon="🛒")

    st.title("🛒 CosmicWorks Chatbot")

    os.environ["OPENAI\_API\_TYPE"] = "azure"

    os.environ["OPENAI\_API\_VERSION"] = os.getenv("AZURE\_OPENAI\_API\_VERSION")

    os.environ["azure\_endpoint"] = os.getenv("AZURE\_OPENAI\_ENDPOINT")

    os.environ["OPENAI\_API\_KEY"] = os.getenv("AZURE\_OPENAI\_API\_KEY")

    os.environ["OPENAI\_EMBEDDINGS\_MODEL\_NAME"] = os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

Save the app.py file

In the command prompt, let’s run the application:

Makre sure you are in the correct virtual environment:

.venv\scripts\activate

Run the application with streamlit run app.py

A close-up of a chatbot

Description automatically generated

Great!

Let’s now add code to our application to run simple queries

Create a main() function

def main():

    st.write("Product embeddings are stored in Azure Cosmos DB for MongoDB vCore")

    st.write("Conversations are stored in Azure Cosmos DB for NoSQL")

*# Set up the LLM*

    llm = AzureChatOpenAI(

        deployment\_name=os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

        temperature=0,

        max\_tokens=1000

    )

*# Set up the LLMChain*

    template = """You are an AI chatbot having a conversation with a human.

    Human: {human\_input}

    AI: """

    prompt = PromptTemplate(input\_variables=["human\_input"], template=template)

    llm\_chain = LLMChain(llm=llm, prompt=prompt)

*# Set up the conversation*

    if prompt := st.chat\_input():

        st.chat\_message("human").write(prompt)

        with st.spinner("Please wait.."):

            response = llm\_chain.run(prompt)

            st.chat\_message("ai").write(response)

Modify the application entry point:

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    main()

Run the application again with streamlit run.app.py

In the textbox, enter “What is the capital of France?” and submit the query

You should get this answer:

A screenshot of a computer

Description automatically generated

Complete code:

import streamlit as st

import os, uuid

from urllib.parse import quote

from datetime import datetime

from langchain.chains import LLMChain

from langchain.prompts import PromptTemplate

from langchain\_openai import AzureChatOpenAI, AzureOpenAIEmbeddings

from langchain.schema import HumanMessage

from langchain.memory import ConversationBufferMemory, CosmosDBChatMessageHistory

from langchain.memory.chat\_message\_histories import StreamlitChatMessageHistory

from langchain.chains import ConversationalRetrievalChain

from langchain.callbacks.base import BaseCallbackHandler

from langchain.vectorstores import AzureCosmosDBVectorSearch

from dotenv import load\_dotenv

def init\_env():

    load\_dotenv()

    st.set\_page\_config(page\_title="CosmicWorks Chatbot", page\_icon="🛒")

    st.title("🛒 CosmicWorks Chatbot")

    os.environ["OPENAI\_API\_TYPE"] = "azure"

    os.environ["OPENAI\_API\_VERSION"] = os.getenv("AZURE\_OPENAI\_API\_VERSION")

    os.environ["azure\_endpoint"] = os.getenv("AZURE\_OPENAI\_ENDPOINT")

    os.environ["OPENAI\_API\_KEY"] = os.getenv("AZURE\_OPENAI\_API\_KEY")

    os.environ["OPENAI\_EMBEDDINGS\_MODEL\_NAME"] = os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

def main():

    st.write("Product embeddings are stored in Azure Cosmos DB for MongoDB vCore")

    st.write("Conversations are stored in Azure Cosmos DB for NoSQL")

*# Set up the LLM*

    llm = AzureChatOpenAI(

        deployment\_name=os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

        temperature=0,

        max\_tokens=1000

    )

*# Set up the LLMChain*

    template = """You are an AI chatbot having a conversation with a human.

    Human: {human\_input}

    AI: """

    prompt = PromptTemplate(input\_variables=["human\_input"], template=template)

    llm\_chain = LLMChain(llm=llm, prompt=prompt)

*# Set up the conversation*

    if prompt := st.chat\_input():

        st.chat\_message("human").write(prompt)

        with st.spinner("Please wait.."):

            response = llm\_chain.run(prompt)

            st.chat\_message("ai").write(response)

def get\_cosmosdb\_mongodb\_connection\_string():

    host = os.getenv('COSMOSDB\_MONGODB\_HOST')

    username = os.getenv('COSMOSDB\_MONGODB\_USERNAME')

    password = os.getenv('COSMOSDB\_MONGODB\_PASSWORD')

    encoded\_password = quote(password, safe='')

    connection\_string = f'mongodb+srv://{username}:{encoded\_password}@{host}/?tls=true&authMechanism=SCRAM-SHA-256&retrywrites=false&maxIdleTimeMS=120000'

    return connection\_string

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    main()

# Ground the chatbot on your data (RAG)

## Connection string

Create a new function get\_ get\_cosmosdb\_mongodb\_connection\_string():

def get\_cosmosdb\_mongodb\_connection\_string():

    host = os.getenv('COSMOSDB\_MONGODB\_HOST')

    username = os.getenv('COSMOSDB\_MONGODB\_USERNAME')

    password = os.getenv('COSMOSDB\_MONGODB\_PASSWORD')

    encoded\_password = quote(password, safe='')

    connection\_string = f'mongodb+srv://{username}:{encoded\_password}@{host}/?tls=true&authMechanism=SCRAM-SHA-256&retrywrites=false&maxIdleTimeMS=120000'

    return connection\_string

## Embeddings

Copy/Paste this code to create a new function to calculate embeddings from a string:

def calculate\_embeddings(query):

    embeddings = AzureOpenAIEmbeddings(

        azure\_deployment=os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL"),

        openai\_api\_version=os.getenv("OPENAI\_API\_VERSION")

    )

    query\_vector = embeddings.embed\_query(query)

    return query\_vector

## Retriever

Copy/paste this code to create a retriever. The retriever uses the text-embedding-ada-002 model to calculate the embeddings and retrieve vectorized information from our Azure Cosmos DB for MongoDB vCore table (database\_team02.products\_team01)

def configure\_retriever():

    connection\_string = get\_cosmosdb\_mongodb\_connection\_string()

    embeddings = AzureOpenAIEmbeddings(

            azure\_deployment=os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL"),

            openai\_api\_version=os.getenv("OPENAI\_API\_VERSION")

    )

    database\_name = os.getenv('COSMOSDB\_MONGODB\_DATABASE')

    products\_collection\_name = os.getenv("COSMOSDB\_MONGODB\_PRODUCTS")

    namespace = f"{database\_name}.{products\_collection\_name}"

    index\_name = products\_collection\_name + "\_vectorindex"

    vector\_store = AzureCosmosDBVectorSearch.from\_connection\_string(

        connection\_string,

        namespace,

        embeddings,

        index\_name=index\_name

    )

    return vector\_store

## Debug functions

We will now create two classes that will help us understand what is happening behind the scene

class StreamHandler(BaseCallbackHandler):

    def \_\_init\_\_(self, container: st.delta\_generator.DeltaGenerator, initial\_text: str = ""):

        self.container = container

        self.text = initial\_text

        self.run\_id\_ignore\_token = None

        self.complete = False  *# Added flag to track completion*

    def on\_llm\_start(self, serialized: dict, prompts: list, \*\*kwargs):

        if prompts[0].startswith("Human"):

            self.run\_id\_ignore\_token = kwargs.get("run\_id")

    def on\_llm\_new\_token(self, token: str, \*\*kwargs) -> None:

        if self.run\_id\_ignore\_token == kwargs.get("run\_id", False):

            return

        self.text += token

        self.container.markdown(self.text)

    def on\_llm\_end(self, response, \*\*kwargs):

        self.complete = True  *# Mark completion*

class PrintRetrievalHandler(BaseCallbackHandler):

    def \_\_init\_\_(self, container):

        self.status = container.status("\*\*Context Retrieval\*\*")

    def on\_retriever\_start(self, serialized: dict, query: str, \*\*kwargs):

        self.status.write(f"\*\*Question:\*\* {query}")

        self.status.update(label=f"\*\*Context Retrieval:\*\* {query}")

    def on\_retriever\_end(self, documents, \*\*kwargs):

        for doc in documents:

            self.status.markdown(doc.page\_content)

        self.status.update(state="complete")

## Retrieval Augmented Generation (RAG)

Now that we have the environment ready, let’s replace our simple chatbot with a chatbot grounded on the Contoso data, stored in Azure Cosmos DB for MongoDB vCore.

Create a rag() function:

def rag():

    msgs = StreamlitChatMessageHistory(key="langchain\_messages")

    memory = ConversationBufferMemory(

        memory\_key="chat\_history",

        return\_messages=True

    )

*# Setup vector store, LLM and QA chain*

    vector\_store = configure\_retriever()

    llm = AzureChatOpenAI(

        deployment\_name=os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

        temperature=0,

        max\_tokens=1000

    )

*# Setup the QA chain*

    qa\_chain = ConversationalRetrievalChain.from\_llm(

        llm,

        retriever=vector\_store.as\_retriever(),

        memory=memory,

        verbose=True

    )

    if len(msgs.messages) == 0:

        msgs.add\_ai\_message("How can I help you?")

    view\_messages = st.expander("View the message contents in session state")

*# Render current messages*

    for msg in msgs.messages:

        st.chat\_message(msg.type).markdown(msg.content, unsafe\_allow\_html=True)

*# If user inputs a new prompt, generate and draw a new response*

    if prompt := st.chat\_input():

        st.chat\_message("human").markdown(prompt, unsafe\_allow\_html=True)

        msgs.add\_user\_message(prompt)

        with st.spinner("Please wait.."):

            retrieval\_handler = PrintRetrievalHandler(st.container())

            stream\_handler = StreamHandler(st.empty())

            response = qa\_chain.run(

                prompt,

                callbacks=[retrieval\_handler, stream\_handler]

            )

            st.chat\_message("ai").markdown(response, unsafe\_allow\_html=True)

            msgs.add\_ai\_message(response)

*# Draw the messages at the end, so newly generated ones show up immediately*

    with view\_messages:

        view\_messages.json(st.session\_state.langchain\_messages)

Modify the application entry point to call our rag() function

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    rag()

Run the application again with streamlit run.app.py

Here are some questions that you can ask:

*What types of bikes do you have?*

*Can you provide more information on mountain bikes?*

*List all types of mountain bikes*

*Can you provide more details on the Mountain-100?*

A screenshot of a chatbot

Description automatically generated

You can expand the dropdown to view the history of messages

A screenshot of a chatbot

Description automatically generated

Full code:

import streamlit as st

import os, uuid

from urllib.parse import quote

from datetime import datetime

from langchain.chains import LLMChain

from langchain.prompts import PromptTemplate

from langchain\_openai import AzureChatOpenAI, AzureOpenAIEmbeddings

from langchain.schema import HumanMessage

from langchain.memory import ConversationBufferMemory, CosmosDBChatMessageHistory

from langchain.memory.chat\_message\_histories import StreamlitChatMessageHistory

from langchain.chains import ConversationalRetrievalChain

from langchain.callbacks.base import BaseCallbackHandler

from langchain.vectorstores import AzureCosmosDBVectorSearch

from dotenv import load\_dotenv

def init\_env():

    load\_dotenv()

    st.set\_page\_config(page\_title="CosmicWorks Chatbot", page\_icon="🛒")

    st.title("🛒 CosmicWorks Chatbot")

    os.environ["OPENAI\_API\_TYPE"] = "azure"

    os.environ["OPENAI\_API\_VERSION"] = os.getenv("AZURE\_OPENAI\_API\_VERSION")

    os.environ["azure\_endpoint"] = os.getenv("AZURE\_OPENAI\_ENDPOINT")

    os.environ["OPENAI\_API\_KEY"] = os.getenv("AZURE\_OPENAI\_API\_KEY")

    os.environ["OPENAI\_EMBEDDINGS\_MODEL\_NAME"] = os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

def main():

    st.write("Product embeddings are stored in Azure Cosmos DB for MongoDB vCore")

    st.write("Conversations are stored in Azure Cosmos DB for NoSQL")

*# Set up the LLM*

    llm = AzureChatOpenAI(

        deployment\_name=os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

        temperature=0,

        max\_tokens=1000

    )

*# Set up the LLMChain*

    template = """You are an AI chatbot having a conversation with a human.

    Human: {human\_input}

    AI: """

    prompt = PromptTemplate(input\_variables=["human\_input"], template=template)

    llm\_chain = LLMChain(llm=llm, prompt=prompt)

*# Set up the conversation*

    if prompt := st.chat\_input():

        st.chat\_message("human").write(prompt)

        with st.spinner("Please wait.."):

            response = llm\_chain.run(prompt)

            st.chat\_message("ai").write(response)

def get\_cosmosdb\_mongodb\_connection\_string():

    host = os.getenv('COSMOSDB\_MONGODB\_HOST')

    username = os.getenv('COSMOSDB\_MONGODB\_USERNAME')

    password = os.getenv('COSMOSDB\_MONGODB\_PASSWORD')

    encoded\_password = quote(password, safe='')

    connection\_string = f'mongodb+srv://{username}:{encoded\_password}@{host}/?tls=true&authMechanism=SCRAM-SHA-256&retrywrites=false&maxIdleTimeMS=120000'

    return connection\_string

def calculate\_embeddings(query):

    embeddings = AzureOpenAIEmbeddings(

        azure\_deployment=os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL"),

        openai\_api\_version=os.getenv("OPENAI\_API\_VERSION")

    )

    query\_vector = embeddings.embed\_query(query)

    return query\_vector

def configure\_retriever():

    connection\_string = get\_cosmosdb\_mongodb\_connection\_string()

    embeddings = AzureOpenAIEmbeddings(

            azure\_deployment=os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL"),

            openai\_api\_version=os.getenv("OPENAI\_API\_VERSION")

    )

    database\_name = os.getenv('COSMOSDB\_MONGODB\_DATABASE')

    products\_collection\_name = os.getenv("COSMOSDB\_MONGODB\_PRODUCTS")

    namespace = f"{database\_name}.{products\_collection\_name}"

    index\_name = products\_collection\_name + "\_vectorindex"

    vector\_store = AzureCosmosDBVectorSearch.from\_connection\_string(

        connection\_string,

        namespace,

        embeddings,

        index\_name=index\_name

    )

    return vector\_store

class StreamHandler(BaseCallbackHandler):

    def \_\_init\_\_(self, container: st.delta\_generator.DeltaGenerator, initial\_text: str = ""):

        self.container = container

        self.text = initial\_text

        self.run\_id\_ignore\_token = None

        self.complete = False  *# Added flag to track completion*

    def on\_llm\_start(self, serialized: dict, prompts: list, \*\*kwargs):

        if prompts[0].startswith("Human"):

            self.run\_id\_ignore\_token = kwargs.get("run\_id")

    def on\_llm\_new\_token(self, token: str, \*\*kwargs) -> None:

        if self.run\_id\_ignore\_token == kwargs.get("run\_id", False):

            return

        self.text += token

        self.container.markdown(self.text)

    def on\_llm\_end(self, response, \*\*kwargs):

        self.complete = True  *# Mark completion*

class PrintRetrievalHandler(BaseCallbackHandler):

    def \_\_init\_\_(self, container):

        self.status = container.status("\*\*Context Retrieval\*\*")

    def on\_retriever\_start(self, serialized: dict, query: str, \*\*kwargs):

        self.status.write(f"\*\*Question:\*\* {query}")

        self.status.update(label=f"\*\*Context Retrieval:\*\* {query}")

    def on\_retriever\_end(self, documents, \*\*kwargs):

        for doc in documents:

            self.status.markdown(doc.page\_content)

        self.status.update(state="complete")

def rag():

    msgs = StreamlitChatMessageHistory(key="langchain\_messages")

    memory = ConversationBufferMemory(

        memory\_key="chat\_history",

        return\_messages=True

    )

*# Setup vector store, LLM and QA chain*

    vector\_store = configure\_retriever()

    llm = AzureChatOpenAI(

        deployment\_name=os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

        temperature=0,

        max\_tokens=1000

    )

*# Setup the QA chain*

    qa\_chain = ConversationalRetrievalChain.from\_llm(

        llm,

        retriever=vector\_store.as\_retriever(),

        memory=memory,

        verbose=True

    )

    if len(msgs.messages) == 0:

        msgs.add\_ai\_message("How can I help you?")

    view\_messages = st.expander("View the message contents in session state")

*# Render current messages*

    for msg in msgs.messages:

        st.chat\_message(msg.type).markdown(msg.content, unsafe\_allow\_html=True)

*# If user inputs a new prompt, generate and draw a new response*

    if prompt := st.chat\_input():

        st.chat\_message("human").markdown(prompt, unsafe\_allow\_html=True)

        msgs.add\_user\_message(prompt)

        with st.spinner("Please wait.."):

            retrieval\_handler = PrintRetrievalHandler(st.container())

            stream\_handler = StreamHandler(st.empty())

            response = qa\_chain.run(

                prompt,

                callbacks=[retrieval\_handler, stream\_handler]

            )

            st.chat\_message("ai").markdown(response, unsafe\_allow\_html=True)

            msgs.add\_ai\_message(response)

*# Draw the messages at the end, so newly generated ones show up immediately*

    with view\_messages:

        view\_messages.json(st.session\_state.langchain\_messages)

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    rag()

# Store conversation history in Azure Cosmos DB for NoSQL

Let’s store the conversations history into an Azure Cosmos DB for NoSQL container.

Create a new method to initialize the NoSQL instance:

def init\_cosmos\_nosql\_history():

    cosmos\_endpoint = f"https://{os.getenv('COSMOSDB\_NOSQL\_ACCOUNT')}.documents.azure.com:443/"

    cosmos\_key = os.getenv('COSMOSDB\_NOSQL\_KEY')

    cosmos\_database = os.getenv('COSMOSDB\_NOSQL\_DATABASE\_NAME')

    cosmos\_container = os.getenv('COSMOSDB\_NOSQL\_CONTAINER\_NAME')

    cosmos\_connection\_string = f"AccountEndpoint={cosmos\_endpoint};AccountKey={cosmos\_key}"

    current\_dt = str(datetime.now().strftime("%Y%m%d\_%H%M%S"))

*# get user\_id from session\_state*

    if "session\_id" not in st.session\_state:

        st.session\_state.session\_id = str(uuid.uuid4())

*# get user\_id from session\_state (in a real app, we would read from authenticated user)*

    if "user\_id" not in st.session\_state:

        st.session\_state.user\_id = str(uuid.uuid4())

    cosmos\_nosql = CosmosDBChatMessageHistory(

        cosmos\_endpoint=cosmos\_endpoint,

        cosmos\_database=cosmos\_database,

        cosmos\_container=cosmos\_container,

        connection\_string=cosmos\_connection\_string,

        session\_id=current\_dt,

        user\_id=st.session\_state.user\_id

    )

*# prepare the cosmosdb instance*

    cosmos\_nosql.prepare\_cosmos()

    return cosmos\_nosql

Duplicate the rag() function and rename the new function to rag\_with\_cosmos\_history()

def rag\_with\_cosmos\_history():

    cosmos\_nosql = init\_cosmos\_nosql\_history()

    msgs = StreamlitChatMessageHistory(key="langchain\_messages")

    memory = ConversationBufferMemory(

        memory\_key="chat\_history",

        chat\_memory=cosmos\_nosql,

        return\_messages=True

    )

*# Setup vector store, LLM and QA chain*

    vector\_store = configure\_retriever()

    llm = AzureChatOpenAI(

        deployment\_name=os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

        temperature=0,

        max\_tokens=1000

    )

*# Setup the QA chain*

    qa\_chain = ConversationalRetrievalChain.from\_llm(

        llm,

        retriever=vector\_store.as\_retriever(),

        memory=memory,

        verbose=True

    )

    if len(msgs.messages) == 0:

        msgs.add\_ai\_message("How can I help you?")

    view\_messages = st.expander("View the message contents in session state")

*# Render current messages*

    for msg in msgs.messages:

        st.chat\_message(msg.type).markdown(msg.content, unsafe\_allow\_html=True)

*# If user inputs a new prompt, generate and draw a new response*

    if prompt := st.chat\_input():

        st.chat\_message("human").markdown(prompt, unsafe\_allow\_html=True)

        msgs.add\_user\_message(prompt)

        with st.spinner("Please wait.."):

            retrieval\_handler = PrintRetrievalHandler(st.container())

            stream\_handler = StreamHandler(st.empty())

            response = qa\_chain.run(

                prompt,

                callbacks=[retrieval\_handler, stream\_handler]

            )

            st.chat\_message("ai").markdown(response, unsafe\_allow\_html=True)

            msgs.add\_ai\_message(response)

*# Draw the messages at the end, so newly generated ones show up immediately*

    with view\_messages:

        view\_messages.json(st.session\_state.langchain\_messages)

Modify the application entry point to call our rag\_with\_cosmos\_history() function

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    rag\_with\_cosmos\_history ()

Run the application again with streamlit run.app.py

Ask a question, such as “Can you provide more details on the Mountain-100?”

# Check history in Azure Cosmos DB for NoSQL

* Go into the Azure Cosmos DB for NoSQL account in the Azure Portal: cosmos-nosql-2024
* Click on “Data Explorer”
* Expand database\_user01 > conversations > Items
* Select the last document

A screenshot of a computer

Description automatically generated

Complete code for this lab

import streamlit as st

import os, uuid

from urllib.parse import quote

from datetime import datetime

from langchain.chains import LLMChain

from langchain.prompts import PromptTemplate

from langchain\_openai import AzureChatOpenAI, AzureOpenAIEmbeddings

from langchain.schema import HumanMessage

from langchain.memory import ConversationBufferMemory, CosmosDBChatMessageHistory

from langchain.memory.chat\_message\_histories import StreamlitChatMessageHistory

from langchain.chains import ConversationalRetrievalChain

from langchain.callbacks.base import BaseCallbackHandler

from langchain.vectorstores import AzureCosmosDBVectorSearch

from dotenv import load\_dotenv

def init\_env():

    load\_dotenv()

    st.set\_page\_config(page\_title=" CosmicWorks Chatbot", page\_icon="🛒")

    st.title("🛒 CosmicWorks Chatbot")

    os.environ["OPENAI\_API\_TYPE"] = "azure"

    os.environ["OPENAI\_API\_VERSION"] = os.getenv("AZURE\_OPENAI\_API\_VERSION")

    os.environ["azure\_endpoint"] = os.getenv("AZURE\_OPENAI\_ENDPOINT")

    os.environ["OPENAI\_API\_KEY"] = os.getenv("AZURE\_OPENAI\_API\_KEY")

    os.environ["OPENAI\_EMBEDDINGS\_MODEL\_NAME"] = os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

def main():

    st.write("Product embeddings are stored in Azure Cosmos DB for MongoDB vCore")

    st.write("Conversations are stored in Azure Cosmos DB for NoSQL")

*# Set up the LLM*

    llm = AzureChatOpenAI(

        deployment\_name=os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

        temperature=0,

        max\_tokens=1000

    )

*# Set up the LLMChain*

    template = """You are an AI chatbot having a conversation with a human.

    Human: {human\_input}

    AI: """

    prompt = PromptTemplate(input\_variables=["human\_input"], template=template)

    llm\_chain = LLMChain(llm=llm, prompt=prompt)

*# Set up the conversation*

    if prompt := st.chat\_input():

        st.chat\_message("human").write(prompt)

        with st.spinner("Please wait.."):

            response = llm\_chain.run(prompt)

            st.chat\_message("ai").write(response)

def get\_cosmosdb\_mongodb\_connection\_string():

    host = os.getenv('COSMOSDB\_MONGODB\_HOST')

    username = os.getenv('COSMOSDB\_MONGODB\_USERNAME')

    password = os.getenv('COSMOSDB\_MONGODB\_PASSWORD')

    encoded\_password = quote(password, safe='')

    connection\_string = f'mongodb+srv://{username}:{encoded\_password}@{host}/?tls=true&authMechanism=SCRAM-SHA-256&retrywrites=false&maxIdleTimeMS=120000'

    return connection\_string

def calculate\_embeddings(query):

    embeddings = AzureOpenAIEmbeddings(

        azure\_deployment=os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL"),

        openai\_api\_version=os.getenv("OPENAI\_API\_VERSION")

    )

    query\_vector = embeddings.embed\_query(query)

    return query\_vector

def configure\_retriever():

    connection\_string = get\_cosmosdb\_mongodb\_connection\_string()

    embeddings = AzureOpenAIEmbeddings(

            azure\_deployment=os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL"),

            openai\_api\_version=os.getenv("OPENAI\_API\_VERSION")

    )

    database\_name = os.getenv('COSMOSDB\_MONGODB\_DATABASE')

    products\_collection\_name = os.getenv("COSMOSDB\_MONGODB\_PRODUCTS")

    namespace = f"{database\_name}.{products\_collection\_name}"

    index\_name = products\_collection\_name + "\_vectorindex"

    vector\_store = AzureCosmosDBVectorSearch.from\_connection\_string(

        connection\_string,

        namespace,

        embeddings,

        index\_name=index\_name

    )

    return vector\_store

class StreamHandler(BaseCallbackHandler):

    def \_\_init\_\_(self, container: st.delta\_generator.DeltaGenerator, initial\_text: str = ""):

        self.container = container

        self.text = initial\_text

        self.run\_id\_ignore\_token = None

        self.complete = False  *# Added flag to track completion*

    def on\_llm\_start(self, serialized: dict, prompts: list, \*\*kwargs):

        if prompts[0].startswith("Human"):

            self.run\_id\_ignore\_token = kwargs.get("run\_id")

    def on\_llm\_new\_token(self, token: str, \*\*kwargs) -> None:

        if self.run\_id\_ignore\_token == kwargs.get("run\_id", False):

            return

        self.text += token

        self.container.markdown(self.text)

    def on\_llm\_end(self, response, \*\*kwargs):

        self.complete = True  *# Mark completion*

class PrintRetrievalHandler(BaseCallbackHandler):

    def \_\_init\_\_(self, container):

        self.status = container.status("\*\*Context Retrieval\*\*")

    def on\_retriever\_start(self, serialized: dict, query: str, \*\*kwargs):

        self.status.write(f"\*\*Question:\*\* {query}")

        self.status.update(label=f"\*\*Context Retrieval:\*\* {query}")

    def on\_retriever\_end(self, documents, \*\*kwargs):

        for doc in documents:

            self.status.markdown(doc.page\_content)

        self.status.update(state="complete")

def rag():

    msgs = StreamlitChatMessageHistory(key="langchain\_messages")

    memory = ConversationBufferMemory(

        memory\_key="chat\_history",

        return\_messages=True

    )

*# Setup vector store, LLM and QA chain*

    vector\_store = configure\_retriever()

    llm = AzureChatOpenAI(

        deployment\_name=os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

        temperature=0,

        max\_tokens=1000

    )

*# Setup the QA chain*

    qa\_chain = ConversationalRetrievalChain.from\_llm(

        llm,

        retriever=vector\_store.as\_retriever(),

        memory=memory,

        verbose=True

    )

    if len(msgs.messages) == 0:

        msgs.add\_ai\_message("How can I help you?")

    view\_messages = st.expander("View the message contents in session state")

*# Render current messages*

    for msg in msgs.messages:

        st.chat\_message(msg.type).markdown(msg.content, unsafe\_allow\_html=True)

*# If user inputs a new prompt, generate and draw a new response*

    if prompt := st.chat\_input():

        st.chat\_message("human").markdown(prompt, unsafe\_allow\_html=True)

        msgs.add\_user\_message(prompt)

        with st.spinner("Please wait.."):

            retrieval\_handler = PrintRetrievalHandler(st.container())

            stream\_handler = StreamHandler(st.empty())

            response = qa\_chain.run(

                prompt,

                callbacks=[retrieval\_handler, stream\_handler]

            )

            st.chat\_message("ai").markdown(response, unsafe\_allow\_html=True)

            msgs.add\_ai\_message(response)

*# Draw the messages at the end, so newly generated ones show up immediately*

    with view\_messages:

        view\_messages.json(st.session\_state.langchain\_messages)

def init\_cosmos\_nosql\_history():

    cosmos\_endpoint = f"https://{os.getenv('COSMOSDB\_NOSQL\_ACCOUNT')}.documents.azure.com:443/"

    cosmos\_key = os.getenv('COSMOSDB\_NOSQL\_KEY')

    cosmos\_database = os.getenv('COSMOSDB\_NOSQL\_DATABASE\_NAME')

    cosmos\_container = os.getenv('COSMOSDB\_NOSQL\_CONTAINER\_NAME')

    cosmos\_connection\_string = f"AccountEndpoint={cosmos\_endpoint};AccountKey={cosmos\_key}"

    current\_dt = str(datetime.now().strftime("%Y%m%d\_%H%M%S"))

*# get user\_id from session\_state*

    if "session\_id" not in st.session\_state:

        st.session\_state.session\_id = str(uuid.uuid4())

*# get user\_id from session\_state (in a real app, we would read from authenticated user)*

    if "user\_id" not in st.session\_state:

        st.session\_state.user\_id = str(uuid.uuid4())

    cosmos\_nosql = CosmosDBChatMessageHistory(

        cosmos\_endpoint=cosmos\_endpoint,

        cosmos\_database=cosmos\_database,

        cosmos\_container=cosmos\_container,

        connection\_string=cosmos\_connection\_string,

        session\_id=current\_dt,

        user\_id=st.session\_state.user\_id

    )

*# prepare the cosmosdb instance*

    cosmos\_nosql.prepare\_cosmos()

    return cosmos\_nosql

def rag\_with\_cosmos\_history():

    cosmos\_nosql = init\_cosmos\_nosql\_history()

    msgs = StreamlitChatMessageHistory(key="langchain\_messages")

    memory = ConversationBufferMemory(

        memory\_key="chat\_history",

        chat\_memory=cosmos\_nosql,

        return\_messages=True

    )

*# Setup vector store, LLM and QA chain*

    vector\_store = configure\_retriever()

    llm = AzureChatOpenAI(

        deployment\_name=os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

        temperature=0,

        max\_tokens=1000

    )

*# Setup the QA chain*

    qa\_chain = ConversationalRetrievalChain.from\_llm(

        llm,

        retriever=vector\_store.as\_retriever(),

        memory=memory,

        verbose=True

    )

    if len(msgs.messages) == 0:

        msgs.add\_ai\_message("How can I help you?")

    view\_messages = st.expander("View the message contents in session state")

*# Render current messages*

    for msg in msgs.messages:

        st.chat\_message(msg.type).markdown(msg.content, unsafe\_allow\_html=True)

*# If user inputs a new prompt, generate and draw a new response*

    if prompt := st.chat\_input():

        st.chat\_message("human").markdown(prompt, unsafe\_allow\_html=True)

        msgs.add\_user\_message(prompt)

        with st.spinner("Please wait.."):

            retrieval\_handler = PrintRetrievalHandler(st.container())

            stream\_handler = StreamHandler(st.empty())

            response = qa\_chain.run(

                prompt,

                callbacks=[retrieval\_handler, stream\_handler]

            )

            st.chat\_message("ai").markdown(response, unsafe\_allow\_html=True)

            msgs.add\_ai\_message(response)

*# Draw the messages at the end, so newly generated ones show up immediately*

    with view\_messages:

        view\_messages.json(st.session\_state.langchain\_messages)

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    rag\_with\_cosmos\_history()