Labs Data & AI Innovation Day

Lab 4: Build a Python chatbot

This document describes how to build a Python chatbot to query the Azure Cosmos DB for NoSQL 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
- An existing Python virtual environment, as described in Lab1

Create a simple chatbot

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

- Create a "**Lab4**" folder in the "**Labs**" foler
- Open Visual Studio Code
- In the "**lab4**" folder, create a file call **simple_chatbot.py**
- Copy/paste the import statements

```
import streamlit as st
import os, time
from openai import AzureOpenAI
from dotenv import load_dotenv
```

- Copy this code to load the content of the .env file

```
def init_env():
    st.set_page_config(page_title="CosmicWorks Chatbot", page_icon="🛒")
    st.title("🛒 CosmicWorks Chatbot")

    load_dotenv("..\.env")

    os.environ["OPENAI_API_TYPE"] = "azure"
    os.environ["OPENAI_API_VERSION"] = os.getenv("AZURE_OPENAI_API_VERSION")
    os.environ["AZURE_OPENAI_ENDPOINT"] = os.getenv("AZURE_OPENAI_ENDPOINT")
    os.environ["AZURE_OPENAI_API_KEY"] = os.getenv("AZURE_OPENAI_API_KEY")
    os.environ["AZURE_OPENAI_EMBEDDING_MODEL"] =
os.getenv("AZURE_OPENAI_EMBEDDING_MODEL")

if __name__ == "__main__":
    init_env()
```

Save the app.py file

- Open a command prompt
- Navigate to the "Lab4" folder
- Activate the virtual environment with "**../.venv/scripts/activate**"
- Run the application with **streamlit run simple_chatbot.py**

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

Add a new function called **`simple_question()`**

```
def simple_question():
    """ Ask a simple question """
    openai_client = AzureOpenAI(
        api_key = os.getenv("AZURE_OPENAI_API_KEY"),
        api_version = os.getenv("AZURE_OPENAI_API_VERSION"),
        azure_endpoint =os.getenv("AZURE_OPENAI_ENDPOINT")
    )

    question = st.text_input("How can I help you?")
    if st.button("Submit"):
        user_prompt = f"""
You are a chatbot, having a friendly conversation with a human
- Answer in markdown format

USER QUESTION:
{question}

ANSWER:
"""
        response, completion_time = get_completion(
            openai_client,
            os.getenv("AZURE_OPENAI_CHAT_MODEL"),
            user_prompt)

        response, completion_time
```

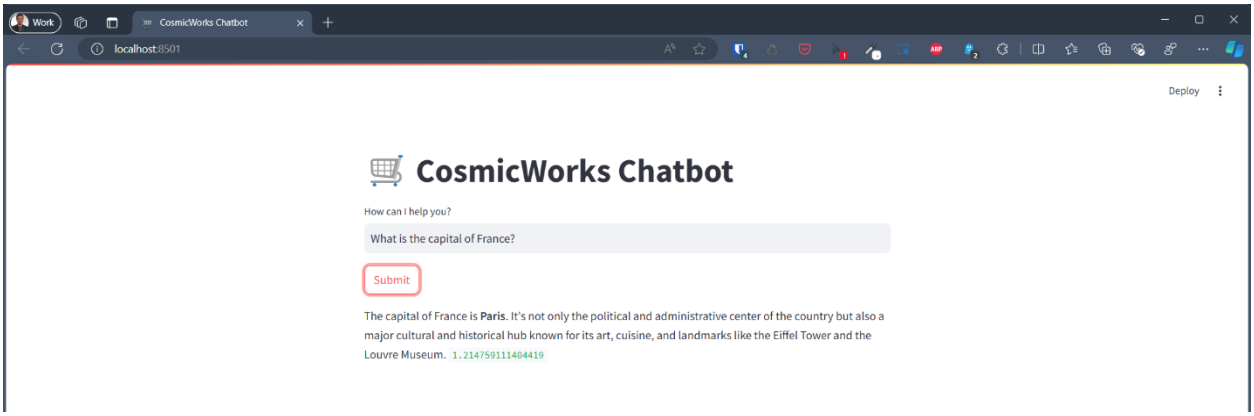
Modify the application entry point:

```
if __name__ == "__main__":
    init_env()
    simple_question()
```

Run the application again with **`streamlit run simple_chatbot.py`**

In the textbox, enter "**`What is the capital of France?`**" and submit the query

You should get this answer:


Work
CosmicWorks Chatbot
localhost:8501
Deploy
CosmicWorks Chatbot
How can I help you?
What is the capital of France?
Submit
The capital of France is Paris. It's not only the political and administrative center of the country but also a major cultural and historical hub known for its art, cuisine, and landmarks like the Eiffel Tower and the Louvre Museum. 1.214759111404419

Complete code:

```
import streamlit as st
import os, time
from openai import AzureOpenAI
from dotenv import load_dotenv

def init_env():
    st.set_page_config(page_title="CosmicWorks Chatbot", page_icon="🛒")
    st.title("🛒 CosmicWorks Chatbot")

    load_dotenv("..\.env")

    os.environ["OPENAI_API_TYPE"] = "azure"
    os.environ["OPENAI_API_VERSION"] = os.getenv("AZURE_OPENAI_API_VERSION")
    os.environ["AZURE_OPENAI_ENDPOINT"] = os.getenv("AZURE_OPENAI_ENDPOINT")
    os.environ["AZURE_OPENAI_API_KEY"] = os.getenv("AZURE_OPENAI_API_KEY")
    os.environ["AZURE_OPENAI_EMBEDDING_MODEL"] =
os.getenv("AZURE_OPENAI_EMBEDDING_MODEL")

def get_completion(openai_client, model, prompt: str):

    start_time = time.time()
    response = openai_client.chat.completions.create(
        model = model,
        messages = [{"role": "user", "content": prompt}]
    )
    end_time = time.time()
    elapsed_time = end_time - start_time
    return response.choices[0].message.content, elapsed_time

def simple_question():
    """ Ask a simple question """
    openai_client = AzureOpenAI(
        api_key = os.getenv("AZURE_OPENAI_API_KEY"),
        api_version = os.getenv("AZURE_OPENAI_API_VERSION"),
        azure_endpoint =os.getenv("AZURE_OPENAI_ENDPOINT")
    )

    question = st.text_input("How can I help you?")
    if st.button("Submit"):
        user_prompt = f"""
You are a chatbot, having a friendly conversation with a human
- Answer in markdown format

USER QUESTION:
{question}
```

```
ANSWER:
"""
        response, completion_time = get_completion(
            openai_client,
            os.getenv("AZURE_OPENAI_CHAT_MODEL"),
            user_prompt)

        response, completion_time

if __name__ == "__main__":
    init_env()
    simple_question()
```

Ground the chatbot on your data (RAG)

In this exercice, we will create a chatbot that can answer questions on the CosmicWorks inventory, stored in the Azure Cosmos DB for NoSQL database.

- In the "**lab4**" folder, create a file call **rag_chatbot**.py
- Copy/paste the import statements (note the additional import statements for Cosmos DB and tenacity)

```
import streamlit as st
import os, time
from azure.cosmos import CosmosClient
from tenacity import retry, wait_random_exponential, stop_after_attempt
from openai import AzureOpenAI
from dotenv import load_dotenv
```

- From the previous exercise, copy the **init_env()** and **get_completion()** code to load the environment variables

```
def init_env():
    st.set_page_config(page_title="CosmicWorks Chatbot", page_icon="🛒")
    st.title("🛒 CosmicWorks Chatbot")

    load_dotenv("..\.env")

    os.environ["OPENAI_API_TYPE"] = "azure"
    os.environ["OPENAI_API_VERSION"] = os.getenv("AZURE_OPENAI_API_VERSION")
    os.environ["AZURE_OPENAI_ENDPOINT"] = os.getenv("AZURE_OPENAI_ENDPOINT")
    os.environ["AZURE_OPENAI_API_KEY"] = os.getenv("AZURE_OPENAI_API_KEY")
    os.environ["AZURE_OPENAI_EMBEDDING_MODEL"] =
os.getenv("AZURE_OPENAI_EMBEDDING_MODEL")

def get_completion(openai_client, model, prompt: str):

    start_time = time.time()
    response = openai_client.chat.completions.create(
        model = model,
        messages = [{"role": "user", "content": prompt}]
    )
    end_time = time.time()
    elapsed_time = end_time - start_time
    return response.choices[0].message.content, elapsed_time
```

- Copy this code to **init_cosmos()**

```
def init_cosmos():
```

```
    COSMOS_DB_ENDPOINT = os.getenv('AZURE_COSMOSDB_NOSQL_ENDPOINT')
    COSMOS_DB_KEY = os.getenv('AZURE_COSMOSDB_NOSQL_KEY')
    DATABASE_NAME = os.getenv('AZURE_COSMOSDB_NOSQL_DATABASE_NAME')
    CONTAINER_NAME = os.getenv('AZURE_COSMOSDB_NOSQL_CONTAINER_NAME')
    client = CosmosClient(COSMOS_DB_ENDPOINT, COSMOS_DB_KEY)
    database = client.get_database_client(DATABASE_NAME)
    products_container = database.get_container_client(CONTAINER_NAME)
    return client, database, products_container
```

- Create a function to create embeddings from a string

```
@retry(wait=wait_random_exponential(min=1, max=20),
stop=stop_after_attempt(10))
def generate_embeddings(openai_client, text):
    """
    Generates embeddings for a given text using the OpenAI API v1.x
    """

    return openai_client.embeddings.create(
        input = text,
        model= os.getenv("AZURE_OPENAI_EMBEDDING_MODEL")
    ).data[0].embedding
```

- Create a method to send a prompt to Azure OpenAI and generate a response

```
def get_completion(openai_client, model, prompt: str):
    start_time = time.time()
    response = openai_client.chat.completions.create(
        model = model,
        messages = [{"role": "user", "content": prompt}]
    )
    end_time = time.time()
    elapsed_time = end_time - start_time
    return response.choices[0].message.content, elapsed_time
```

- Create a method to query the Cosmos DB database to get a list of documents similar to the user's question

```
def get_similar_docs(openai_client, container, query_text, limit=5):
    """
        Get similar documents from Cosmos DB for NoSQL

        input:
            container: name of the container
            query_text: user question
            limit: max number of documents to return
        output:
            documents: json documents similar to the user question
            elapsed_time
```

```
    """
    # vectorize the question
    query_vector = generate_embeddings(openai_client, query_text)

    # find product keys of products that match the question
    query = f"""
        SELECT TOP {limit}
            VALUE c.productKey
        FROM c
        WHERE c.type = 'vector'
        ORDER BY VectorDistance(c.embedding, {query_vector})
    """
    start_time = time.time()

    results = container.query_items(
        query=query,
        parameters=None,
        enable_cross_partition_query=True
    )

    # get products from list of id
    id_list = [id for id in results]

    id_list_str = ', '.join([f"'{id}'" for id in id_list])
    query = f"""
        SELECT * FROM c
        WHERE c.type = 'product' AND c.id IN ({id_list_str})
    """
    results = container.query_items(
        query=query,
        enable_cross_partition_query=True
    )

    products = []
    for product in results:
        products.append(product)

    end_time = time.time()
    elapsed_time = end_time - start_time

    return products, elapsed_time
```

- And finally, create the **`main()`** entry point for the application

```
def main():
    """ Use vector search in Cosmos DB for NoSQL to answer the question """
    top_k = 10
```

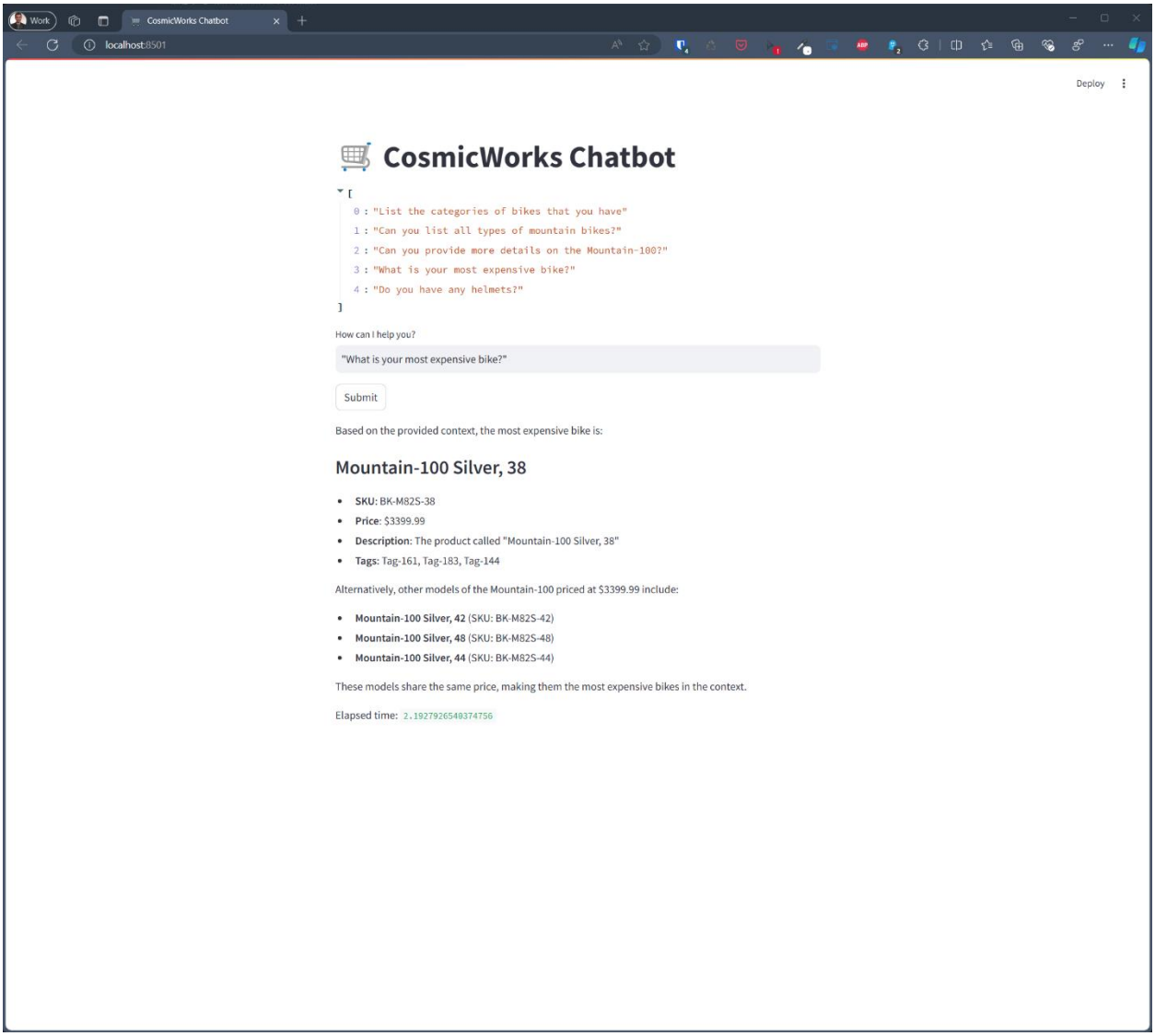
```
    questions = [
        "List the categories of bikes that you have",
        "Can you list all types of mountain bikes?",
        "Can you provide more details on the Mountain-100?",
        "What is your most expensive bike?",
        "Do you have any helmets?"
    ]
    questions
    question = st.text_input("How can I help you?")

    # init Cosmos DB and Azure OpenAI
    client, database, container = init_cosmos()
    openai_client = AzureOpenAI(
        api_key = os.getenv("AZURE_OPENAI_API_KEY"),
        api_version = os.getenv("AZURE_OPENAI_API_VERSION"),
        azure_endpoint =os.getenv("AZURE_OPENAI_ENDPOINT")
    )

    if st.button("Submit"):
        with st.spinner("Please wait.."):

            # get similar docs from Cosmos DB
            docs, elapsed_time = get_similar_docs(openai_client, container,
question, top_k)

            # pass docs in the context and generate response using Azure
OpenAI
            user_prompt = f"""
    Using the following CONTEXT, answer the user's question as best as
possible.
    - Answer in English
    - Answer in markdown format

    CONTEXT:
    {docs}

    USER QUESTION:
    {question}

    ANSWER:
    """
            response, completion_time = get_completion(openai_client,
os.getenv("AZURE_OPENAI_CHAT_MODEL"), user_prompt)
            st.write(response, unsafe_allow_html = True)
            st.write("Elapsed time:", completion_time)

if __name__ == "__main__":
    init_env()
```

```
    main()
```

- Start the application with **`streamlit run rag_chatbot.py`**

Here are some questions that you can ask:

```
List the categories of bikes that you have
Can you list all types of mountain bikes?
Can you provide more details on the Mountain-100?
What is your most expensive bike?
Do you have any helmets?
```



You can expand the dropdown to view the history of messages

Full code:

```
import streamlit as st
import os, time
from azure.cosmos import CosmosClient
from tenacity import retry, wait_random_exponential, stop_after_attempt
from openai import AzureOpenAI
from dotenv import load_dotenv

def init_env():
    st.set_page_config(page_title="CosmicWorks Chatbot", page_icon="🛒")
    st.title("🛒 CosmicWorks Chatbot")

    load_dotenv("..\.env")

    os.environ["OPENAI_API_TYPE"] = "azure"
    os.environ["OPENAI_API_VERSION"] = os.getenv("AZURE_OPENAI_API_VERSION")
    os.environ["AZURE_OPENAI_ENDPOINT"] = os.getenv("AZURE_OPENAI_ENDPOINT")
    os.environ["AZURE_OPENAI_API_KEY"] = os.getenv("AZURE_OPENAI_API_KEY")
    os.environ["AZURE_OPENAI_EMBEDDING_MODEL"] =
os.getenv("AZURE_OPENAI_EMBEDDING_MODEL")

def get_completion(openai_client, model, prompt: str):

    start_time = time.time()
    response = openai_client.chat.completions.create(
        model = model,
        messages = [{"role": "user", "content": prompt}]
    )
    end_time = time.time()
    elapsed_time = end_time - start_time
    return response.choices[0].message.content, elapsed_time

def init_cosmos():
    COSMOS_DB_ENDPOINT = os.getenv('AZURE_COSMOSDB_NOSQL_ENDPOINT')
    COSMOS_DB_KEY = os.getenv('AZURE_COSMOSDB_NOSQL_KEY')
    DATABASE_NAME = os.getenv('AZURE_COSMOSDB_NOSQL_DATABASE_NAME')
    CONTAINER_NAME = os.getenv('AZURE_COSMOSDB_NOSQL_CONTAINER_NAME')
    client = CosmosClient(COSMOS_DB_ENDPOINT, COSMOS_DB_KEY)
    database = client.get_database_client(DATABASE_NAME)
    products_container = database.get_container_client(CONTAINER_NAME)
    return client, database, products_container

@retry(wait=wait_random_exponential(min=1, max=20),
stop=stop_after_attempt(10))
def generate_embeddings(openai_client, text):
    """
    Generates embeddings for a given text using the OpenAI API v1.x
```

```
    """

    return openai_client.embeddings.create(
        input = text,
        model= os.getenv("AZURE_OPENAI_EMBEDDING_MODEL")
    ).data[0].embedding

def get_completion(openai_client, model, prompt: str):
    start_time = time.time()
    response = openai_client.chat.completions.create(
        model = model,
        messages = [{"role": "user", "content": prompt}]
    )
    end_time = time.time()
    elapsed_time = end_time - start_time
    return response.choices[0].message.content, elapsed_time

def get_similar_docs(openai_client, container, query_text, limit=5):
    """
        Get similar documents from Cosmos DB for NoSQL

        input:
            container: name of the container
            query_text: user question
            limit: max number of documents to return
        output:
            documents: json documents similar to the user question
            elapsed_time
    """
    # vectorize the question
    query_vector = generate_embeddings(openai_client, query_text)

    # find product keys of products that match the question
    query = f"""
        SELECT TOP {limit}
            VALUE c.productKey
        FROM c
        WHERE c.type = 'vector'
        ORDER BY VectorDistance(c.embedding, {query_vector})
    """
    start_time = time.time()

    results = container.query_items(
        query=query,
        parameters=None,
        enable_cross_partition_query=True
    )
```

```
    # get products from list of id
    id_list = [id for id in results]

    id_list_str = ', '.join([f"'{id}'" for id in id_list])
    query = f"""
        SELECT * FROM c
        WHERE c.type = 'product' AND c.id IN ({id_list_str})
    """
    results = container.query_items(
        query=query,
        enable_cross_partition_query=True
    )

    products = []
    for product in results:
        products.append(product)

    end_time = time.time()
    elapsed_time = end_time - start_time

    return products, elapsed_time

def main():
    """ Use vector search in Cosmos DB for NoSQL to answer the question """
    top_k = 10

    questions = [
        "List the categories of bikes that you have",
        "Can you list all types of mountain bikes?",
        "Can you provide more details on the Mountain-100?",
        "What is your most expensive bike?",
        "Do you have any helmets?"
    ]
    questions
    question = st.text_input("How can I help you?")

    # init Cosmos DB and Azure OpenAI
    client, database, container = init_cosmos()
    openai_client = AzureOpenAI(
        api_key = os.getenv("AZURE_OPENAI_API_KEY"),
        api_version = os.getenv("AZURE_OPENAI_API_VERSION"),
        azure_endpoint =os.getenv("AZURE_OPENAI_ENDPOINT")
    )

    if st.button("Submit"):
        with st.spinner("Please wait.."):

            # get similar docs from Cosmos DB
```

```
            docs, elapsed_time = get_similar_docs(openai_client, container,
question, top_k)

            # pass docs in the context and generate response using Azure OpenAI
            user_prompt = f"""
    Using the following CONTEXT, answer the user's question as best as
possible.
    - Answer in English
    - Answer in markdown format

    CONTEXT:
    {docs}

    USER QUESTION:
    {question}

    ANSWER:
    """
            response, completion_time = get_completion(openai_client,
os.getenv("AZURE_OPENAI_CHAT_MODEL"), user_prompt)
            st.write(response, unsafe_allow_html = True)
            st.write("Elapsed time:", completion_time)

if __name__ == "__main__":
    init_env()
    main()
```