I have an existing app.py that is used to manage my front end for dancescoop. The code is as follows.  
  
# app.py

import streamlit as st

import requests

import os

import yaml

import logging

from dotenv import load\_dotenv

# Set up basic logging

logging.basicConfig(level=logging.INFO)

logging.info("app.py: Streamlit app starting...")

# Load environment variables and configuration

load\_dotenv()

# Calculate the base directory and config path

base\_dir = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

config\_path = os.path.join(base\_dir, 'config', 'config.yaml')

# Load YAML configuration

with open(config\_path, "r") as f:

    config = yaml.safe\_load(f)

logging.info("app.py: config completed.")

# Get FastAPI API URL from environment variable

FASTAPI\_API\_URL = os.getenv("FASTAPI\_API\_URL", "https://social-dance-app-ws-main.onrender.com/query")

if not FASTAPI\_API\_URL:

    raise ValueError("The environment variable FASTAPI\_API\_URL is not set.")

st.set\_page\_config(layout="wide")

# Initialize the chat message history in session state

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

    st.session\_state["messages"] = []

# Load chatbot instructions from a file specified in the YAML config

instructions\_path = os.path.join(base\_dir, config['prompts']['chatbot\_instructions'])

with open(instructions\_path, "r") as file:

    chatbot\_instructions = file.read()

st.markdown(chatbot\_instructions)

def error\_handling(e, custom\_message=None):

    """

    Handle errors by appending a standardized error message to the chat history.

    If custom\_message is provided, it will be used as the first line of the error message.

    """

    if custom\_message:

        error\_message = (

            f"{custom\_message} "

            "I can answer questions such as:\n\n"

            "1. Where can I dance salsa tonight?\n"

            "2. Where can I dance tango this month? Only show me the social dance events.\n"

            "3. When does the West Coast Swing event on Saturdays start?\n"

            "4. etc. etc. ..."

        )

    else:

        error\_message = (

            "Sorry, I did not quite catch that.\n\n"

            "I can answer questions such as:\n\n"

            "1. Where can I dance salsa tonight?\n"

            "2. Where can I dance tango this month? Only show me the social dance events.\n"

            "3. When does the West Coast Swing event on Saturdays start?\n"

            "4. etc. etc. ..."

        )

    st.session\_state["messages"].append({"role": "assistant", "content": error\_message})

    logging.error(f"app.py: Error encountered - {e}")

# Get user input and send it to the FastAPI backend

user\_input = st.text\_area("Ask a question, then click Send:", height=100)

if st.button("Send"):

    if user\_input.strip():

        # Display the user's message in the chat history

        st.session\_state["messages"].append({"role": "user", "content": user\_input})

        logging.info("app.py: About to send user input to FastAPI backend.")

        try:

            # Send the query to the FastAPI backend

            response = requests.post(FASTAPI\_API\_URL, json={"user\_input": user\_input})

            response.raise\_for\_status()

            data = response.json()

            # Get the event data from the response

            events = data["data"]

            if events:

                # Create a scrollable container to hold the events

                with st.container():

                    st.markdown("<hr>", unsafe\_allow\_html=True)  # Add a separator line

                    for event in events:

                        event\_name = event.get('event\_name', 'No Name')

                        url = event.get('url', '#')

                        # Only create a hyperlink if the URL is properly formatted (starts with "http")

                        if isinstance(url, str) and url.startswith("http"):

                            st.markdown(f'<a href="{url}" target="\_blank"><strong>{event\_name}</strong></a>', unsafe\_allow\_html=True)

                        else:

                            st.markdown(f"\*\*{event\_name}\*\*")

                        # Display other event details (one row per column)

                        for column\_name, value in event.items():

                            if column\_name not in ('event\_name', 'url'):

                                st.markdown(f"\*\*{column\_name}\*\*: {value}")

                        st.markdown("<hr>", unsafe\_allow\_html=True)

            else:

                # If no events are returned and a valid SQL query exists, call error\_handling with a custom message BEFORE showing the SQL query.

                if data.get('sql\_query'):

                    error\_handling("No events returned", custom\_message="Sorry, I could not find those events in my database.")

            # Display the SQL query (shown after error handling if triggered)

            st.markdown(f"\*\*SQL Query\*\*:\n```\n{data.get('sql\_query', 'No SQL query provided')}\n```")

        except Exception as e:

            error\_handling(e)

    else:

        st.write("Please enter a message")

else:

    st.write("Please enter a message")

# Render the conversation history from newest to oldest without a header

for message in reversed(st.session\_state["messages"]):

    if message["role"] == "user":

        st.markdown(f"\*\*You wrote:\*\* {message['content']}")

    else:

        st.markdown(message["content"])

I want to improve this code so that it follows this behavior according to the following questions (Q) and answers (A). The questions are asked by the chatbot. The answers are what we expect from the user. The structure of this prompt is I will explain the context of the question as C1, C2, etc. Q1, Q2, etc. will be the actual question that will be asked and A1, A2, etc. will be at least 1 example answer that can be expected. E1, E2, etc. will be an explanation if required for you to generate the code and the prompts.

You will need to generate working code as well as the prompts required to make this work. I also want you to keep the context of each interaction with each user so they do not need to ask the same questions over and over again.

C1. This is the starting point. If they know how to use the system, they can just ask a question. If not, we go thru a question and answer process whereby we get them to formulate a question that DanceScoop can answer efficiently and effectively for the user. In the case of the A1 answer being Yes, they might type Yes and then their query or simply a query. If they enter a query that can not be generated to create SQL that returns a valid answer, then you will put them thru the A1 No process.

Q1. Hi, I am the DanceScoop chatbot. I will help you query the database to get the answers back that you want. If you have used DanceScoop before, just answer “Yes, then provide your query and I will not ask you any more questions.”

A1. Yes

E1. Simply leave the interface as you see it now. They can enter a query in the chatbot interface and that will use an existing prompt (prompts/chatbot\_instructions.txt) to generate a SQL statement that is submitted to Postgres. For this branch, you are done. The present logic is fine.

A1. No

E1. Continue on with this next part.

Q2. Pick one of these event\_type(s).

1. Social dance
2. Dance classes
3. Dance workshops
4. Live music venues where you can partner dance
5. 5. Other

A2. social dance

E2. They should choose a number or numbers or they can write out. Valid answers would be: social dance, 1, dance classes, 2, …

C2. So, a valid answer for Q2 would be cha cha, double shuffle, rueda, or swing for example. The point is we want the specific dance\_style(s). They can select more than one.

Q2. Please choose one of these dance\_style(s).

1. What dance\_style(s). Give them this list.
   1. Ballroom
      1. cha cha, foxtrot, quickstep, rumba, waltz
   2. Country
      1. 2 step, double shuffle, line dancing, nite club 2
   3. Latin
      1. bachata, cumbia, kizomba, merengue, rueda, salsa
   4. Swing
      1. balboa, lindy, swing, wcs, west coast swing

A2. rueda, salsa

C3. We need a time frame that they are interested in. Frame the question in terms of what you already know from above.

Q3. When do you want to social dance salsa or rueda?

A3. This week

E3. They can answer any time frame as long as it is no more than 3 days in the past or 9 months in the future.

C4. Now we should write the question out for them that we know DanceScoop can answer.

Q4. Please confirm that this is what you want to know? “Where can I social dance salsa or rueda this week?”

A4. Yes

C4. Submit the Q4 question using the (prompts/chatbot\_instructions.txt) to generate the sql.

A4. No

C4. Please have them correct the question. Provide them this feedback. “Sorry I got that wrong. Can you look at this question and tell me what I need to change? Valid questions are phrasaes like: “Where can I social dance tonight? What time do the salsa events start on the weekend? At the minimum we need a time frame. However, if you can include the dance\_style(s) and the event\_type(s) you will get a much better answer.