```
import os
import json
import logging

# Flask
from flask import Flask, request
from flask_socketio import SocketIO, emit
from engineio.async_drivers import eventlet

# Langchain
from langchain_core.documents import Document
from langchain_core.tools import tool
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_core.runnables import Runnable
from langchain_core.runnables.history import RunnableWithMessageHistory
from langchain_core.chat_history import BaseChatMessageHistory

from langchain.agents import AgentExecutor, create_tool_calling_agent

from langchain_community.chat_message_histories import ChatMessageHistory
from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder

# Embedding
from langchain_community.vectorstores import FAISS
from langchain_community.retrievers import BM25Retriever

from langchain.retrievers import EnsembleRetriever
from langchain.pydantic_v1 import BaseModel, Field
from langchain_community.embeddings import GPT4AllEmbeddings

from langchain_openai import ChatOpenAI

# OpenStreetMap
from geopy.geocoders import Nominatim

logging.basicConfig(level=logging.DEBUG)

# Environment setup
# Need to use .env files for security
os.environ["OPENAI_API_KEY"] = ""



# Location setup (current)
council_list = {
    'The Berri Barmera Council': 'Berri Barmera Council',
    'The Regional Council of Goyder': 'Regional Council of Goyder',
    'The District Council of Ceduna': 'District Council of Ceduna',
    'The District Council of Coober Pedy': 'District Council of Coober Pedy',
    'The District Council of Franklin Harbour': 'District of Franklin Harbour',
    'The City of Norwood Payneham and St Peters': 'Norwood Payneham and St Peters',
    'The District Council of Kimba': 'District Council of Kimba',
    'The District Council of Loxton Waikerie': 'Loxton Waikerie',
    'The District Council of Peterborough': 'Peterborough',
    'Southern Mallee District Council': 'Southern Mallee',
    'Municipal Council of Roxby Downs': 'Roxby Downs',
    'The District Council of Streaky Bay': 'District Council of Streaky Bay',
    'The District Council of Tumby Bay': 'District Council of Tumby Bay',
    'Pastoral Unincorporated Area': 'Outback Communities Authority',
    'Anangu Pitjantjatjara Yankunytjatjara': 'Outback Communities Authority',
    'The District Council of Yankalilla': 'District Council of Yankalilla',
    'Adelaide City Council': 'City of Adelaide',
    'The District Council of Mount Remarkable': 'Mount Remarkable',
    'The District Council of Cleve': 'District Council of Cleve',
    'The District Council of Lower Eyre Peninsula': 'Lower Eyre Peninsula',
    'The District Council of Karoonda East Murray': 'Karoonda East Murray',
    'The Rural City of Murray Bridge': 'Rural City of Murray Bridge',
    'The District Council of Orroroo Carrieton': 'Orroroo Carrieton',
    'The District Council of Grant': 'District Council of Grant',
    'The Corporation of the City of Whyalla': 'City of Whyalla',
    'The Corporation of the Town of Walkerville': 'Town of Walkerville'
```

```
}
geolocator = Nominatim(user_agent='chatbot_geocoder')

def get_location(address):
    address = address + ', SA, AU'
    location = geolocator.geocode(address, addressdetails=True)

    suburb = ''
    council = ''

    if location:
        addr = location.raw['address']

        # Suburb
        if 'suburb' in addr:
            suburb = addr['suburb']
        elif 'town' in addr:
            suburb = addr['town']
        elif 'hamlet' in addr:
            suburb = addr['hamlet']
        elif 'village' in addr:
            suburb = addr['village']
        elif 'city_district' in addr:
            suburb = addr['city_district']
        elif 'city' in addr:
            suburb = addr['city']

        # Council
        if 'county' in addr:
            council = addr['county']
        elif 'municipality' in addr:
            council = addr['municipality']
        elif 'natural' in addr:
            council = addr['natural']

        if council in council_list:
            council = council_list[council]

    return suburb, council



# VectorDB setup
# embedding (GPT4all local embeddings)
embeddings = GPT4AllEmbeddings()
# FAISS retriever from local database
faiss=FAISS.load_local("faiss_index", embeddings, allow_dangerous_deserialization=True)
# BM25 retriever
docs=[]
with open('sacommunity.json', 'r') as f:
    doc_list = json.load(f)
for i in range(len(doc_list)):
    page=Document(page_content = doc_list[i])
    docs.append(page)
bm25_retriever = BM25Retriever.from_documents(docs, k=5)



# LLM setup (GPT3.5)
# Trial API keys: limited to 100 API calls per minute
llm = ChatOpenAI(model="gpt-3.5-turbo-0125", temperature=0)

# Extract Class tool
class Extract(BaseModel):
    """Extract search term to search a database of communities, organisations or services in South
Australia.
    Examples: financial assistance, football club, footy, soccer club, quit smoking, food assistance
...
    """

    ser: str = Field(
        description="Communities, organisations or services that the customer is interested in,
```

```
without the location.",
    )

    loc: str = Field(None,
        description="The location of the request if provided.",
    )



# Similarity Search function/tool
@tool (args_schema=Extract) #to use output from Extract tool
def search_database(ser: str, loc: str = "") -> str:
    """Run similarity search in vector database to get the community, organisation or service
informations."""

    # If no service, ask user what they want
    if not ser:
        return "Tell user to input what service they want."

    # If no location, perform similarity search on data
    if not loc or loc.lower() == 'sa':
        # FAISS filter not set
        faiss_retriever = faiss.as_retriever(search_kwargs={'k': 10})
        # ensemble retriever
        ensemble_retriever = EnsembleRetriever(retrievers=[bm25_retriever, faiss_retriever], weights=
[0.5, 0.5])

        docs = ensemble_retriever.invoke(ser)
        search_results = []
        for doc in docs:
            search_results.append(doc.page_content)
        if not search_results:
            return "No relevant organisation found, tell user to rephrase question."
        return "\n\n".join(search_results)

    # If given location, get the suburb and council from the given location
    sub, coun = get_location(loc)
    if sub:
        # 1. search for Suburb
        # FAISS filter set to location
        faiss_retriever = faiss.as_retriever(search_kwargs={'filter': dict(location=sub.lower()),
'k': 10, 'fetch_k': 4000})
        # ensemble retriever
        ensemble_retriever = EnsembleRetriever(retrievers=[bm25_retriever, faiss_retriever], weights=
[0.5, 0.5])

        docs = ensemble_retriever.invoke(ser)
        search_results = [f"{loc} suburb is {sub}, council is {coun}."]
        for doc in docs:
            org = doc.page_content
            if 'S Suburb' not in org:
                org += f'\nS Suburb: {sub.title()}\nCouncil: {coun.title()}'
            search_results.append(org)

        # 2. search for Council
        if len(search_results) <= 20:
            # FAISS filter set to council
            faiss_retriever = faiss.as_retriever(search_kwargs={'filter': dict(council=coun.lower()),
'k': 10, 'fetch_k': 4000})

            docs = faiss_retriever.invoke(ser)
            for doc in docs:
                org = doc.page_content
                if 'S Suburb' not in org:
                    org += f'\nS Suburb: {sub.title()}\nCouncil: {coun.title()}'
                search_results.append(org)

        return "\n\n".join(search_results)

    elif coun:
        # FAISS filter set to council
        faiss_retriever = faiss.as_retriever(search_kwargs={'filter': dict(council=coun.lower()),
```

```
'k': 10, 'fetch_k': 4000})
        # ensemble retriever
        ensemble_retriever = EnsembleRetriever(retrievers=[bm25_retriever, faiss_retriever], weights=
[0.5, 0.5])

        docs = ensemble_retriever.invoke(ser)
        search_results = [f"{loc} council is {coun}."]
        for doc in docs:
            org = doc.page_content
            if 'S Suburb' not in org:
                org += f'\nCouncil: {coun.title()}'
            search_results.append(org)

        return "\n\n".join(search_results)

    else:
        return f"{loc} is not a correct address, tell user to check their location."



# Agent setup
system="""You are the friendly customer service assistant for SAcommunity.org, a real website
dedicated to helping users find organizations or communities that provide the assistance they need.
        Step 1, use the tool when needed to extract keyword for similarity search to obtain the
relevant organizations.
        Step 2, from the result, recommend organisations based on the 'Services' and 'Subjects' they
offer, relevant to user query.
        Step 3, from all relevant organisations, prioritize recommendations based on location,
indicated in the 'S Suburb' or 'Council' fields of organisations.
        Step 4, list out all relevant organizations, strictly using the following format (fill in the
relevant "Org Name" and "Org ID"):
        • "Org Name" https://sacommunity.org/org/"Org ID"
        If no relevant organisation found, do not invent or speculate on responses.
        """
prompt = ChatPromptTemplate.from_messages(
    [
        ("system",system),
        MessagesPlaceholder(variable_name='chat_history'),
        ("user", "{input}"),
        MessagesPlaceholder(variable_name="agent_scratchpad"),
    ]
)
# Construct the tool calling agent
agent = create_tool_calling_agent(llm, [search_database], prompt)
agent_executor: Runnable = AgentExecutor(agent=agent, tools=[search_database], verbose=True)


# Chat history setup
store = {} #to store each clients session id and chat history

def get_session_history(session_id: str) -> BaseChatMessageHistory:
    if session_id not in store:
        store[session_id] = ChatMessageHistory()
    return store[session_id]

message_history = ChatMessageHistory()
agent_with_chat_history = RunnableWithMessageHistory(
    agent_executor,
    get_session_history,
    input_messages_key="input",
    history_messages_key="chat_history",
)




# Connection setup
app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins="*", async_mode='eventlet', logger=True)

@socketio.on('connect')
def handle_connect():
```

```
    print('Client connected')
     # Generate a unique session ID for the client
    session_id = str(request.sid)  # Use request.sid from Flask-SocketIO
    print(session_id)
    emit('status', {'message': 'Connected to server'})

@socketio.on('disconnect')
def handle_disconnect():
    del store[str(request.sid)]
    print('Client disconnected')

#an event handler that gets triggered when the server receives a 'user_query' event from a client via
WebSocket
@socketio.on('user_query')
def handle_user_query(data):
    try:
        session_id = str(request.sid) #Again, request.sid is used to retrieve the session ID of the
client who sent the message.
        user_input = data['userInput']
        result = agent_with_chat_history.invoke(
            {"input": user_input},
            config={"configurable": {"session_id": session_id}},
        )
        emit('chat_response', {'reply': result['output']})
    except Exception as e:
        logging.error(f"Error handling user query: {str(e)}", exc_info=True)
        emit('error', {'error': str(e)})

if __name__ == '__main__':
    socketio.run(app, host='0.0.0.0', port=5000, debug=True)
```