```python
from dotenv import load_dotenv
import os
import json
import logging
import requests
from datetime import datetime
import pytz
from langchain_openai import ChatOpenAI
from langchain_core.messages import HumanMessage, SystemMessage, ToolMessage,
AIMessage
from langchain_core.tools import tool
# Agent 활용 : 3개 이상의 많은 도구를 사용할 때 효율적
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_google_community import GoogleSearchAPIWrapper
# 설치 필요 : pip install langchain-google-community

# 로깅 설정
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

load_dotenv()

# 단일 도구: get_weather_info
@tool
def get_current_weather(location: str) -> str:
    """
    주어진 장소의 날씨 정보를 조회합니다.
    장소명을 받아서 자동으로 좌표를 조회한 후 날씨를 조회합니다.

    Args:
        location: 장소명 (예: 'LA', '서울', 'Paris', '도쿄')

    Returns:
        포맷된 날씨 정보 문자열 (장소, 기온, 체감기온, 습도, 바람, 날씨)
    """
    try:
        #  Step 1: 장소명으로 좌표 조회 (Nominatim API)
        logger.info(f"Step 1: '{location}'의 좌표 조회 중...")

        coord_url =
f"https://nominatim.openstreetmap.org/search?q={location}&format=json"
        coord_headers = {
            "User-Agent": "LangChainWeatherBot/1.0 (weather@bot.com)"
        }

        coord_response = requests.get(coord_url, headers=coord_headers, timeout=5)
        coord_response.raise_for_status()
        coord_results = coord_response.json()
```

```
        if not coord_results:
            logger.warning(f"'{location}'의 좌표를 찾을 수 없습니다.")
            return f"'{location}'의 좌표를 찾을 수 없습니다."

        latitude = float(coord_results[0]['lat'])
        longitude = float(coord_results[0]['lon'])
        logger.info(f"✅ 좌표 조회 성공: {location} → 위도={latitude}, 경도={longitude}")

        # Step 2: 좌표로 날씨 정보 조회 (OpenWeatherMap API)
        logger.info(f"Step 2: 날씨 정보 조회 중...")

        api_key = os.getenv("OPENWEATHER_API_KEY")
        weather_url = (
            f"http://api.openweathermap.org/data/2.5/weather?"
            f"lat={latitude}&lon={longitude}&appid={api_key}&units=metric&lang=kr"
        )

        weather_response = requests.get(weather_url, timeout=5)
        weather_response.raise_for_status()
        weather_data = weather_response.json()

        # API 키 유효성 검증
        if weather_data.get('cod') == '401':
            logger.error("API 키가 유효하지 않습니다.")
            return "API 키가 유효하지 않습니다."

        # 날씨 데이터 추출
        if 'weather' not in weather_data:
            logger.warning("날씨 정보를 찾을 수 없습니다.")
            return "날씨 정보를 불러올 수 없습니다."

        # 날씨 상세 정보 추출
        weather_desc = weather_data['weather'][0]['description']
        main_weather = weather_data['weather'][0]['main']  # 'Clear', 'Clouds', 'Rain'
등


        # 기온 정보
        temp = weather_data['main']['temp']
        feels_like = weather_data['main']['feels_like']
        temp_min = weather_data['main']['temp_min']
        temp_max = weather_data['main']['temp_max']

        # 습도 및 바람
        humidity = weather_data['main']['humidity']
        pressure = weather_data['main']['pressure']
        wind_speed = weather_data['wind'].get('speed', 'N/A')
        wind_deg = weather_data['wind'].get('deg', 'N/A')

        # 구름 및 가시거리
        clouds = weather_data.get('clouds', {}).get('all', 'N/A')
```

```
        visibility = weather_data.get('visibility', 'N/A')

        # 장소명 (API 응답에서 얻는 공식 이름)
        location_name = weather_data.get('name', location)
        country = weather_data.get('sys', {}).get('country', '')

        logger.info(f"✅ 날씨 정보 조회 성공: {location_name}")

        # 결과 포맷팅
        result = (
            f"장소: {location_name}{f' ({country})' if country else ''}\n"
            f"————————————————————\n"
            f"현재 기온: {temp}°C\n"
            f"체감 기온: {feels_like}°C\n"
            f"기온 범위: {temp_min}°C ~ {temp_max}°C\n"
            f"습도: {humidity}%\n"
            f"기압: {pressure} hPa\n"
            f"바람: {wind_speed}m/s"
        )

        if wind_deg != 'N/A':
            result += f" ({wind_deg}°)"

        result += (
            f"\n날씨: {weather_desc} ({main_weather})\n"
            f"구름: {clouds}%\n"
            f"가시거리: {visibility}m"
        )

        return result

    except requests.exceptions.Timeout:
        logger.error(f"'{location}' 조회 중 요청 시간 초과")
        return f"'{location}' 조회 중 요청 시간 초과. 잠시 후 다시 시도해주세요."
    except requests.exceptions.RequestException as e:
        logger.error(f"네트워크 오류: {e}")
        return f"네트워크 오류가 발생했습니다: {str(e)}"
    except (ValueError, KeyError, IndexError) as e:
        logger.error(f"데이터 파싱 오류: {e}")
        return f"날씨 정보를 처리하는 중 오류가 발생했습니다."
    except Exception as e:
        logger.error(f"예상치 못한 오류: {e}", exc_info=True)
        return f"오류가 발생했습니다: {str(e)}"

# 날짜와 시간
@tool
def get_current_datetime(timezone: str, location: str) -> str:
    """ 현재 시각을 반환하는 함수

    Args:
        timezone (str): 타임존 (예: 'Asia/Seoul') 실제 존재하는 타임존이어야 함
```

```
        location (str): 지역명. 타임존이 모든 지명에 대응되지 않기 때문에 이후 llm 답변 생성에 사용됨
    """
    tz = pytz.timezone(timezone)
    now = datetime.now(tz).strftime("%Y-%m-%d %H:%M:%S")
    location_and_local_time = f'{timezone} ({location}) 현재시각 {now} '
    # 타임존, 지역명, 현재시각을 문자열로 반환
    print(location_and_local_time)
    return location_and_local_time

@tool
def google_search(query: str) -> str:
    """
    구글 검색을 통해 필요한 정보를 검색합니다.
    Args:
        query: 검색어
    """
    try:
        search = GoogleSearchAPIWrapper(google_api_key=os.getenv("GOOGLE_API_KEY"))
        result = search.run(query)
        return result
    except Exception as e:
        return f"검색 중 오류 발생: {str(e)}"


@tool
def calculate(expression: str) -> str:
    """
    수학 계산을 수행합니다.
    Args:
        expression: 계산식 (예: "5 + 3 * 2")
    """
    try:
        # 안전한 계산 : builtins 내장함수 사용 제한(os 명령등 실행하면 위험)
        result = eval(expression, {"__builtins__": {}})
        return f"계산 결과: {expression} = {result}"
    except Exception as e:
        return f"계산 중 오류 발생: {str(e)}"


# 에이전트 설정

import os
from datetime import datetime
from langchain.agents import create_agent
from langchain_openai import ChatOpenAI
from langchain_core.tools import tool


# LLM 초기화
llm = ChatOpenAI(
```

```
    model="gpt-4.1-mini",
    temperature=0,
)

# 도구 목록
tools = [
    get_current_datetime,
    get_current_weather,
    google_search,
    calculate,
]

# 에이전트 생성 (1.0 방식)
agent = create_agent(
    model=llm,
    tools=tools,
    system_prompt="""당신은 여행 및 일상의 계획에 매우 도움이 되는 어시스턴트입니다.
사용자의 요청에 따라 다음의 도구들을 활용하여 답변합니다.
1. 현재 날짜 및 시간 조회
2. 실시간 날씨 조회
3. 구글 검색을 통한 정보 검색
4. 수학 계산

사용자의 질문을 분석하여 가장 적절한 도구를 선택하고 조합하여 사용하세요.
한국어로 친절하고 상세하게 답변하세요.
여러 도구가 필요하면 순차적으로 사용하세요."""
)

# 실행 함수
def run_agent(user_input: str):
    """에이전트 실행"""
    print(f"\n{'='*70}")
    print(f"👤 사용자: {user_input}")
    print(f"{'='*70}")

    try:
        # 1.0 방식: messages 형식으로 invoke
        response = agent.invoke({
            "messages": [{"role": "user", "content": user_input}]
        })
        print(f"\n{'='*70}")
        # print(f"🤖 어시스턴트:\n{response}")
        print(f"{'='*70}\n")

        final_answer = response['messages'][-1].content
        return final_answer

    except Exception as e:
        print(f"❌ 에러 발생: {str(e)}\n")
        return None
```

```python
# 테스트
if __name__ == "__main__":
    # 테스트 케이스들

    # 1. 시간 조회
    final_answer=run_agent("지금 현재 시간이 몇 시야?")
    print(f'💬 : {final_answer}')
    # 2. 날씨 조회
    # run_agent("서울의 날씨는 어떻게 돼?")

    # 3. 검색
    final_answer=run_agent("AI Agent 가 무엇인지 한 줄로 알려줘")
    print(f'💬 : {final_answer}')
    # 4. 복합 질문 (여러 도구 조합)
    # run_agent("지금 시간이 몇 시고, 부산의 날씨는 어떻게 되는지 알려주세요")

    # 5. 정보 검색과 날씨
    final_answer=run_agent("충북 괴산의 날씨를 알려주고, 2025년 11월 괴산의 축제 일정을
알려줘")
    print(f'💬 : {final_answer}')
```