**C.E. en IA y BD – PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL**

**1. DESCRIPCIÓN DEL RETO**

En este reto educativo, desarrollarás un asistente virtual con temática futbolística, enfocado en el torneo de fútbol sala de "La Liga Solidaria FP Cantabria". El asistente proporcionará información sobre los equipos participantes, reglas del torneo, horarios de partidos, estiramientos recomendados y acciones solidarias relacionadas con el evento.

**2. MISIÓN**

Diseñar y programar un asistente virtual que interactúe con el usuario, respondiendo a consultas específicas sobre el torneo de fútbol sala, fomentando la participación, el conocimiento del evento y promoviendo el espíritu solidario entre los participantes.

**3. RELACIÓN CON LOS CONTENIDOS DEL MÓDULO “PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL”**

Este reto se alinea con los contenidos del módulo de **Programación de Inteligencia Artificial**, específicamente con los siguientes resultados de aprendizaje y criterios de evaluación:

* **RA1**: Caracteriza lenguajes de programación valorando su idoneidad en el desarrollo de Inteligencia Artificial.

c) Se ha determinado el lenguaje de programación más apropiado para el desarrollo de la aplicación de Inteligencia Artificial.

d) Se han valorado características de los lenguajes de programación para el desarrollo de Inteligencia Artificial.

* **RA2**: Desarrolla aplicaciones de Inteligencia Artificial utilizando entornos de modelado.

c) Se ha definido el modelo que se quiere implementar según el problema planteado.

d) Se ha implementado la aplicación de Inteligencia Artificial.

e) Se han evaluado los resultados obtenidos.

**4. MOTIVACIÓN**

Este reto te permitirá aplicar los conocimientos adquiridos en programación e inteligencia artificial en un contexto real y cercano. Además, contribuirás al proyecto solidario "La Liga Solidaria FP Cantabria", promoviendo el espíritu deportivo y la solidaridad en la comunidad educativa.

**5. OBJETIVOS**

* Aplicar técnicas de programación para desarrollar un asistente virtual temático.
* Integrar y manipular datos estructurados (JSON) para proporcionar respuestas precisas.
* Desarrollar habilidades en procesamiento del lenguaje natural.
* Fomentar el trabajo en equipo y la colaboración.
* Contribuir al proyecto solidario difundiendo información relevante.

**6. RELACIÓN CON EL PROYECTO**

Este reto es parte integral del "Proyecto de Innovación La Liga Solidaria FP Cantabria", apoyando tanto su componente deportivo como solidario. Al crear un asistente virtual especializado, facilitarás el acceso a información clave del torneo y promoverás las acciones solidarias asociadas.

Este asistente servirá de apoyo para aquellas personas en situaciones de dependencia que quieran aprender más sobre el deporte y no puedan leer contenido en los sistemas convencionales, ya que utiliza la voz para comunicarse.

**7. EL RETO**

**7.1 INTRODUCCIÓN**

En esta actividad, mejorarás un asistente virtual existente, añadiéndole nuevas funciones centradas en el torneo de fútbol sala. Ten en cuenta que esta actividad está diseñada para practicar las instrucciones y métodos vistos en clase, por lo que puede no ser la forma óptima de desarrollar un programa con la misma funcionalidad.

**7.2 PARTE I: CREACIÓN DEL ASISTENTE BÁSICO [REALIZADO EN CLASE]**

Antes de añadir nuevas funcionalidades, vamos a construir el asistente virtual desde cero. Este asistente podrá escuchar tu voz, procesar el audio y responder mediante voz sintetizada.

**Paso 1: Importar Librerías**

Comenzaremos importando las librerías necesarias para nuestro proyecto.

**Código:**

# 1. Importar las librerías recientemente instaladas

import pyttsx3 # Transformar texto a voz

import speech\_recognition as sr # Reconocer la voz del usuario

import pywhatkit # Realizar búsquedas en la web y YouTube

import yfinance as yf # Ver los precios de las acciones (opcional)

import pyjokes # Obtener chistes en diferentes idiomas

# 2. Importar librerías ya incluidas en Python

import webbrowser # Manejar el navegador

import datetime # Trabajar con fechas y horas

import wikipedia # Recibir información de Wikipedia

import json # Manejar archivos JSON

import random # Generar respuestas aleatorias

from difflib import get\_close\_matches # Para coincidencias aproximadas

**Instrucciones:**

* Asegúrate de tener instaladas las librerías externas utilizando pip install nombre\_libreria si es necesario.
* Importa todas las librerías al inicio de tu script para tenerlas disponibles en todo el código.

**Paso 2: Función para Transformar Audio en Texto**

Esta función permitirá al asistente escuchar a través del micrófono y convertir lo que escuche en texto.

**Código:**

def transformar\_audio\_en\_texto():

# Almacenar el recognizer en una variable

r = sr.Recognizer()

# Configurar el micrófono

with sr.Microphone() as microfono:

# Añadir un tiempo de espera

r.pause\_threshold = 1

# Informar que comienza la grabación

print("Ya puedes hablar.")

# Guardar lo que se escuche como audio

audio = r.listen(microfono)

try:

# Utilizar el reconocimiento de voz de Google

texto = r.recognize\_google(audio, language="es-es")

# Mostrar el texto en pantalla

print("Dijiste: " + texto)

# Devolver el texto

return texto

except sr.UnknownValueError:

# Error si no entiende el audio

print("Ups, no te he entendido.")

return "sigo esperando"

except sr.RequestError:

# El servicio está inactivo

print("Ups, el servicio está inactivo.")

return "sigo esperando"

except:

# Cualquier otro error

print("Ups, algo ha salido mal.")

return "sigo esperando"

**Instrucciones:**

* Utiliza la librería speech\_recognition para capturar audio desde el micrófono.
* Implementa manejo de excepciones para diferentes tipos de errores que puedan ocurrir durante el reconocimiento de voz.

**Paso 3: Reconocer las Voces Disponibles en el Sistema**

Antes de que el asistente pueda hablar, necesitamos saber qué voces están disponibles en nuestro sistema.

**Código:**

# Script para obtener los IDs de las voces disponibles

engine = pyttsx3.init()

for voz in engine.getProperty('voices'):

print(voz)

**Instrucciones:**

* Ejecuta este script para listar las voces disponibles en tu sistema.
* Observa los IDs y selecciona la voz que prefieras para tu asistente.
* Comenta o elimina este script después de obtener los IDs, ya que solo es necesario para la configuración inicial.

**Paso 4: Función para que el Asistente Hable**

Ahora crearemos la función que permitirá al asistente sintetizar voz a partir de texto.

**Código:**

id1 = "HKEY\_LOCAL\_MACHINE\\SOFTWARE\\Microsoft\\Speech\\Voices\\Tokens\\TTS\_MS\_ES-ES\_HELENA\_11.0"

# Asegúrate de reemplazar el ID con la voz que hayas seleccionado

def hablar(mensaje, id):

# Inicializar el motor de pyttsx3

engine = pyttsx3.init()

# Seleccionar la voz

engine.setProperty('voice', id)

# Configurar propiedades adicionales

engine.setProperty('rate', 150) # Velocidad del habla

engine.setProperty('volume', 1) # Volumen de la voz

# Encolar el mensaje

engine.say(mensaje)

# Ejecutar y esperar a que termine

engine.runAndWait()

**Instrucciones:**

* Utiliza la librería pyttsx3 para convertir texto en voz.
* Configura la voz, velocidad y volumen según tus preferencias.
* Prueba la función con mensajes de ejemplo para asegurarte de que funciona correctamente.

**Paso 5: Función de Saludo Inicial**

Crearemos una función que permita al asistente saludar al usuario dependiendo de la hora del día.

**Código:**

def saludo\_inicial():

# Obtener la hora actual

hora = datetime.datetime.now()

if hora.hour < 6 or hora.hour > 20:

momento = "Buenas noches."

elif 6 <= hora.hour < 13:

momento = "Buenos días."

else:

momento = "Buenas tardes."

# Saludo personalizado

hablar(momento + " Soy tu asistente virtual. ¿Cómo te puedo ayudar?", id1)

**Instrucciones:**

* Utiliza el módulo datetime para obtener la hora actual.
* Define diferentes saludos basados en la hora del día.
* Asegúrate de que el asistente pronuncie el saludo utilizando la función hablar.

**Paso 6: Funciones Adicionales (Hora y Día)**

El asistente podrá informar al usuario sobre la fecha y la hora actuales.

**Código para Informar del Día:**

def pedir\_dia():

# Obtener la fecha de hoy

dia = datetime.date.today()

# Obtener el número del día de la semana (lunes = 0, domingo = 6)

dia\_semana = dia.weekday()

# Diccionario con los nombres de los días

calendario = {

0: "lunes",

1: "martes",

2: "miércoles",

3: "jueves",

4: "viernes",

5: "sábado",

6: "domingo"

}

hablar(f'Hoy es {calendario[dia\_semana]}', id1)

**Código para Informar de la Hora:**

def pedir\_hora():

hora = datetime.datetime.now()

hora\_formateada = f'En este momento son las {hora.hour} horas, {hora.minute} minutos y {hora.second} segundos.'

hablar(hora\_formateada, id1)

**Instrucciones:**

* En pedir\_dia(), utiliza el método weekday() para obtener el día de la semana y mapea el número al nombre del día.
* En pedir\_hora(), formatea la hora actual para que sea fácilmente comprensible.
* Utiliza la función hablar para que el asistente comunique la información al usuario.

**Paso 7: Función Central del Asistente**

Crearemos la función principal pedir\_cosas() que coordinará las interacciones con el usuario.

**Código:**

def pedir\_cosas():

# Activar el saludo inicial

saludo\_inicial()

# Variable de control

comenzar = True

# Variable para el primer saludo

primera\_vez = True

# Bucle principal

while comenzar:

if not primera\_vez:

hablar("¿Necesitas algo más?", id1)

primera\_vez = False

# Escuchar el pedido del usuario

pedido = transformar\_audio\_en\_texto().lower()

# Procesar el pedido

if "qué día es hoy" in pedido:

pedir\_dia()

continue

elif "qué hora es" in pedido:

pedir\_hora()

continue

elif "no gracias" in pedido or "gracias" in pedido:

hablar("¡Hasta la próxima!", id1)

break

else:

hablar("Lo siento, no puedo ayudar con eso.", id1)

**Instrucciones:**

* Implementa un bucle while para mantener la interacción continua con el usuario.
* Añade condiciones para procesar diferentes tipos de pedidos.
* Incluye una opción para terminar la interacción cuando el usuario diga "no gracias" o "gracias".

**Paso 8: Ejecutar el Asistente**

Finalmente, llamamos a la función principal para iniciar el asistente.

**Instrucciones:**

* Coloca esta llamada al final de tu script para que el asistente comience a ejecutarse.
* Asegúrate de que todas las funciones estén definidas antes de esta llamada.

Ahora que tenemos nuestro asistente básico funcionando, estamos listos para añadirle nuevas funcionalidades específicas relacionadas con el torneo de fútbol sala en las siguientes secciones.

**7.3 PARTE II: ASISTENTE DE INFORMACIÓN DE EQUIPOS**

La primera función que añadirás a tu asistente es la capacidad de proporcionar información sobre los equipos participantes.

Imagina que el usuario pregunta:

* **"¿Quiénes forman parte del equipo Tigres FC?"**

Añade un nuevo elif a tu método pedir\_cosas:

**Código:**

elif 'equipo' in pedido:

obtener\_informacion\_equipo(pedido)

continue

Ahora, completa la función obtener\_informacion\_equipo siguiendo el script a continuación y rellenando los huecos:

**Código:**

def obtener\_informacion\_equipo(pedido):

# Extraer el nombre del equipo del pedido

equipo\_nombre = pedido.split('equipo', 1)[1].strip().lower()

# Abrir el archivo JSON con la información de los equipos

with open('equipos.json', 'r', encoding='utf-8') as archivo\_equipos:

equipos = json.load(archivo\_equipos)

# Buscar el equipo en los datos

for equipo in equipos['equipos']:

if equipo['nombre'].lower() == equipo\_nombre:

participantes = ', '.join(equipo['participantes'])

hablar(f"El equipo {equipo['nombre']} está compuesto por: {participantes}.", id1)

return

hablar("Lo siento, no tengo información sobre ese equipo.", id1)

**Instrucciones:**

1. Extrae el nombre del equipo desde el pedido del usuario.
2. Abre el archivo equipos.json y carga los datos.
3. Recorre la lista de equipos y busca el que coincide con el nombre extraído.
4. Si lo encuentras, extrae los participantes y utilízalos para generar la respuesta.
5. Si no encuentras el equipo, informa al usuario.

**7.4 PARTE III: ASISTENTE DE INFORMACIÓN DE HORARIOS**

El asistente informará sobre los horarios y lugares de los partidos.

Ejemplo de pregunta:

* **"¿Cuándo juega el equipo Tigres FC contra Leones FC?"**

Añade el siguiente elif en pedir\_cosas:

**Código:**

elif 'partido' in pedido:

obtener\_informacion\_partido(pedido)

continue

Completa la función obtener\_informacion\_partido:

**Código:**

def obtener\_informacion\_partido(pedido):

# Extraer los nombres de los equipos

equipos\_mencionados = pedido.split('entre', 1)[1].strip().lower()

equipos = equipos\_mencionados.split('y')

if len(equipos) == 2:

equipo1 = equipos[0].strip()

equipo2 = equipos[1].strip()

# Abrir el archivo JSON con los partidos

with open('partidos.json', 'r', encoding='utf-8') as archivo\_partidos:

partidos = json.load(archivo\_partidos)

# Buscar el partido en los datos

for partido in partidos['partidos']:

if (partido['equipo1'].lower() == equipo1 and partido['equipo2'].lower() == equipo2) or \

(partido['equipo1'].lower() == equipo2 and partido['equipo2'].lower() == equipo1):

hablar(f"El partido entre {partido['equipo1']} y {partido['equipo2']} se jugará el {partido['fecha']} a las {partido['hora']} en {partido['lugar']}.", id1)

return

hablar("Lo siento, no tengo información sobre ese partido.", id1)

**Instrucciones:**

1. Extrae los nombres de los equipos del pedido del usuario.
2. Abre el archivo partidos.json y carga los datos.
3. Busca el partido que involucra a ambos equipos.
4. Si lo encuentras, proporciona la información detallada.
5. Si no, informa al usuario de que no tienes esa información.

**7.5 PARTE IV: ASISTENTE DE INFORMACIÓN DE REGLAS**

El asistente proporcionará información sobre las reglas del torneo.

Ejemplo de pregunta:

* **"Explícame la regla de los cambios ilimitados."**

Añade en pedir\_cosas:

**Código:**

elif 'regla' in pedido:

            respuesta = obtener\_informacion\_regla(pedido)

            if respuesta:

                hablar(respuesta, id1)

            else:

                hablar("Lo siento, no tengo una respuesta para eso.", id1)

Completa la función obtener\_informacion\_regla:

**Código:**

def obtener\_informacion\_regla(mensaje):

    with open('UD3\_Prueba\_1/reglas.json', 'r', encoding='utf-8') as archivo:

        datos = json.load(archivo)

    hablar('¿Sobre qué regla quieres preguntar?', id1)

    pedido = transformar\_audio\_en\_texto().lower()

    mensaje = pedido.lower()

    for intent in datos['intents']:

        for pattern in intent['patterns']:

            if pattern.lower() in mensaje:

                return random.choice(intent['responses'])

    return None

**Instrucciones:**

1. Extrae el título de la regla solicitada desde el pedido.
2. Abre el archivo reglas.json y carga los datos.
3. Recorre las reglas y busca la que coincide con el título extraído.
4. Si la encuentras, proporciona su descripción.
5. Si no, informa al usuario.

**7.6 PARTE V: ASISTENTE DE INFORMACIÓN DE SALUD**

El asistente ofrecerá recomendaciones sobre estiramientos y cuidados de salud.

Ejemplo de solicitud:

* **"Recomiéndame estiramientos para después del partido."**

Añade en pedir\_cosas:

**Código:**

elif 'estiramientos' in pedido:

obtener\_recomendaciones\_estiramientos()

continue

Completa la función obtener\_recomendaciones\_estiramientos:

**Código:**

def obtener\_recomendaciones\_estiramientos():

# Abrir el archivo JSON con las recomendaciones

with open('estiramientos.json', 'r', encoding='utf-8') as archivo\_estiramientos:

estiramientos = json.load(archivo\_estiramientos)

# Proporcionar las recomendaciones

for ejercicio in estiramientos['ejercicios']:

hablar(f"{ejercicio['nombre']}: {ejercicio['descripcion']}", id1)

**Instrucciones:**

1. Abre el archivo estiramientos.json y carga los datos.
2. Recorre la lista de ejercicios y proporciona sus nombres y descripciones al usuario.

**7.7 PARTE VI: ASISTENTE INTERACTIVO**

Para todos los demás casos, el asistente responderá según los "patterns" y "responses" de un fichero JSON.

Añade al final de pedir\_cosas:

**Código:**

else:

respuesta = obtener\_respuesta(pedido)

if respuesta:

hablar(respuesta, id1)

else:

hablar("Lo siento, no tengo una respuesta para eso.", id1)

Completa la función obtener\_respuesta basada en la Actividad 07:

**Código:**

import json

import random

from difflib import get\_close\_matches

def obtener\_respuesta(mensaje):

with open('datos.json', 'r', encoding='utf-8') as archivo:

datos = json.load(archivo)

mensaje = mensaje.lower()

for intent in datos['intents']:

for pattern in intent['patterns']:

if pattern.lower() in mensaje:

return random.choice(intent['responses'])

return None

**Instrucciones:**

1. Importa las librerías necesarias (json, random, get\_close\_matches).
2. Abre el fichero datos.json y carga los datos.
3. Recorre los patrones y busca coincidencias con el mensaje del usuario.
4. Si encuentras una coincidencia, devuelve una respuesta aleatoria de las disponibles.
5. Si no, devuelve None.

**Consideraciones Finales**

* Asegúrate de importar todas las librerías necesarias al inicio de tu script, incluyendo las que permiten manejar archivos JSON y procesar texto.
* Prueba tu asistente con diferentes preguntas para verificar que responde correctamente en cada caso.
* Ten en cuenta que los archivos JSON (equipos.json, partidos.json, reglas.json, estiramientos.json, solidario.json, datos.json) deben estar correctamente formateados y ubicados en el mismo directorio que tu script.

**7.8 PARTE VII: ASISTENTE CLIMÁTICO**

El asistente ofrecerá recomendaciones sobre estiramientos y cuidados de salud.

**Paso 1: Importar las Librerías Necesarias**

Primero, debemos importar la librería requests para realizar solicitudes a la API del clima. Si aún no la tienes instalada, puedes hacerlo ejecutando:

pip install requests

Luego, en la sección de importaciones al inicio de tu script, añade:

**Código:**

import requests

**Paso 2: Añadir el elif Correspondiente en pedir\_cosas()**

En la función principal pedir\_cosas(), debemos añadir un nuevo elif para manejar cuando el usuario pregunte por el clima. Coloca este elif en la sección donde procesas el pedido del usuario.

elif "clima" in pedido or "tiempo" in pedido:

informar\_clima(pedido)

continue

**Explicación:**

* Añadimos un elif que verifica si las palabras "clima" o "tiempo" están en el pedido.
* Si es así, llamamos a la función informar\_clima(pedido) y continuamos el bucle.

**Paso 3: Definir la Función informar\_clima(pedido)**

Ahora, crearemos la función que utilizará una API para obtener el clima de la ciudad indicada por el usuario y avisar si hay alguna alerta por mal clima.

**Código:**

def informar\_clima(pedido):

try:

# Preguntar al usuario por la ciudad si no está especificada

if 'en' in pedido:

ciudad = pedido.split('en', 1)[1].strip()

else:

hablar("Por favor, dime la ciudad para consultar el clima.", id1)

ciudad = transformar\_audio\_en\_texto().lower()

# Tu clave de API de OpenWeatherMap (debes obtener tu propia clave)

api\_key = 'TU\_API\_KEY\_AQUÍ' # Reemplaza con tu propia API key

# Construir la URL de la solicitud

url = f"http://api.openweathermap.org/data/2.5/weather?q={ciudad}&appid={api\_key}&lang=es&units=metric"

# Realizar la solicitud a la API

respuesta = requests.get(url)

datos = respuesta.json()

if datos['cod'] == 200:

# Obtener los datos del clima

temp = datos['main']['temp']

descripcion = datos['weather'][0]['description']

mensaje = f"El clima en {ciudad} es de {temp} grados Celsius con {descripcion}."

# Verificar si hay alertas

if 'alerts' in datos:

alerta = datos['alerts'][0]['description']

mensaje += f" Atención: hay una alerta de clima adverso. {alerta}"

# Decir el clima

hablar(mensaje, id1)

else:

hablar(f"No pude obtener el clima para {ciudad}. Por favor verifica el nombre.", id1)

except Exception as e:

print(f"Error al consultar el clima: {e}")

hablar("Hubo un problema al obtener el clima. Inténtalo de nuevo más tarde.", id1)

**Instrucciones:**

1. **Obtener la Ciudad:**
   * Si el usuario menciona la ciudad en su pedido (ejemplo: "¿Cómo está el clima en Madrid?"), extraemos el nombre de la ciudad.
   * Si no menciona la ciudad, le pedimos que nos la diga y utilizamos transformar\_audio\_en\_texto() para capturar su respuesta.
2. **Configurar la API:**
   * Necesitas obtener una clave de API de OpenWeatherMap. Regístrate en OpenWeatherMap para obtener una clave gratuita.
   * Reemplaza 'TU\_API\_KEY\_AQUÍ' con tu clave real.
3. **Realizar la Solicitud a la API:**
   * Construimos la URL con el nombre de la ciudad y nuestra clave de API.
   * Realizamos una solicitud GET y almacenamos la respuesta en datos.
4. **Procesar la Respuesta:**
   * Si el código de respuesta es 200, significa que la solicitud fue exitosa.
   * Obtenemos la temperatura y la descripción del clima.
   * Verificamos si hay alertas de clima en los datos.
     + **Nota:** La API gratuita de OpenWeatherMap no proporciona alertas por defecto. Para acceder a alertas meteorológicas, es posible que necesites utilizar el servicio One Call API y tener una suscripción adecuada.
     + Para este ejemplo, asumiremos que la información de alertas está disponible en datos['alerts'].
   * Formamos el mensaje a comunicar al usuario.
5. **Manejo de Errores:**
   * Si ocurre algún error (por ejemplo, ciudad no encontrada o problemas con la conexión), informamos al usuario.

**Paso 4: Actualizar las Importaciones y Notas**

**Importaciones:**

Asegúrate de que la librería requests esté importada al inicio del script:

**Código:**

import requests # Para realizar solicitudes HTTP a la API del clima

**Notas Importantes:**

* **Obtención de la API Key:**
  + Debes obtener tu propia API Key de OpenWeatherMap.
  + Regístrate en OpenWeatherMap y sigue las instrucciones para obtener una clave gratuita.
* **Limitaciones de la API Gratuita:**
  + La API gratuita puede tener limitaciones en cuanto a la cantidad de solicitudes y los datos disponibles.
  + Para acceder a alertas de clima, es posible que necesites utilizar el servicio One Call API.
  + Revisa la documentación de OpenWeatherMap para más detalles.

**Paso 5: Prueba de la Funcionalidad**

Para asegurarte de que todo funciona correctamente:

1. **Ejecuta tu script.**
2. **Realiza una consulta al asistente:**
   * Di: "¿Cómo está el clima en [ciudad]?" o "¿Qué tiempo hace en [ciudad]?"
3. **Verifica la Respuesta:**
   * El asistente debería proporcionarte la temperatura actual y una breve descripción del clima en la ciudad indicada.
   * Si hay alguna alerta (dependiendo de la API y de la ciudad), también debería mencionarla.

**7.9 PARTE VII: ASISTENTE DE ALERTAS CLIMÁTICAS [OPCIONAL]**

Podemos agregar un hilo (thread) en nuestro asistente virtual que se ejecute en segundo plano y verifique periódicamente si hay alertas meteorológicas en una ciudad específica. Esto permitirá que el asistente continúe interactuando con el usuario mientras el hilo de fondo monitorea las alertas cada X segundos (por ejemplo,300 segundos o 5 minutos). Si detecta alguna alerta, el asistente interrumpirá lo que esté haciendo para informar al usuario.

**Paso 1: Importar las Librerías Necesarias**

Asegúrate de que las siguientes librerías están importadas al inicio de tu script:

**Código:**

import threading

import time

import requests # Si no está ya importada

**Paso 2: Modificar la Función informar\_clima(pedido)**

Vamos a modificar la función informar\_clima(pedido) para que, una vez que el asistente obtenga la ciudad del usuario, inicie el hilo de monitorización de alertas que se ejecutará en segundo plano.

**Código:**

# Iniciar el hilo de monitorización de alertas

hilo\_alertas = threading.Thread(target=monitorear\_alertas, args=(ciudad,))

hilo\_alertas.daemon = True # Para que el hilo se detenga cuando el programa principal termine

hilo\_alertas.start()

**Explicación:**

* Iniciar el Hilo de Monitorización:
  + Después de obtener y comunicar el clima actual, iniciamos el hilo hilo\_alertas que ejecuta la función monitorear\_alertas(ciudad).
  + Establecemos daemon=True para que el hilo no bloquee la finalización del programa principal.

**Paso 3: Definir la Función monitorear\_alertas(ciudad)**

Esta función se ejecutará en segundo plano y revisará las alertas cada 5 minutos.

**Código:**

def monitorear\_alertas(ciudad):

while True:

try:

# Tu clave de API de OpenWeatherMap

api\_key = 'TU\_API\_KEY\_AQUÍ' # Reemplaza con tu propia API key

# Obtener las coordenadas de la ciudad

url\_ciudad = f"http://api.openweathermap.org/data/2.5/weather?q={ciudad}&appid={api\_key}&lang=es&units=metric"

respuesta\_ciudad = requests.get(url\_ciudad)

datos\_ciudad = respuesta\_ciudad.json()

if datos\_ciudad['cod'] == 200:

lat = datos\_ciudad['coord']['lat']

lon = datos\_ciudad['coord']['lon']

# URL para la One Call API (que incluye alertas)

url\_alertas = f"https://api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&appid={api\_key}&lang=es&units=metric"

respuesta\_alertas = requests.get(url\_alertas)

datos\_alertas = respuesta\_alertas.json()

# Verificar si hay alertas

if 'alerts' in datos\_alertas and datos\_alertas['alerts']:

for alerta in datos\_alertas['alerts']:

evento = alerta.get('event', 'Alerta meteorológica')

descripcion = alerta.get('description', '')

mensaje\_alerta = f"¡Atención! Hay una {evento} en {ciudad}. {descripcion}"

# Interrumpir y notificar al usuario

hablar(mensaje\_alerta, id1)

else:

print(f"No hay alertas en {ciudad} en este momento.")

else:

print(f"No se pudo obtener información para la ciudad {ciudad}.")

except Exception as e:

print(f"Error al monitorear alertas: {e}")

# Esperar 5 minutos antes de la siguiente verificación

time.sleep(300) # 300 segundos = 5 minutos

**Explicación:**

* Bucle Infinito:
  + La función utiliza un while True para ejecutar indefinidamente mientras el programa esté corriendo.
* Verificación de Alertas:
  + Cada 5 minutos, hace una solicitud a la API para obtener alertas en la ciudad especificada.
  + Si hay alertas, utiliza la función hablar para informar al usuario, interrumpiendo cualquier otra actividad.
* Interrupción de la Actividad Actual:
  + La llamada a hablar notificará al usuario de inmediato sobre la alerta.

**8. MEJORA EL ASISTENTE**

En este apartado mejoraremos y optimizaremos las funcionalidades del asistente virtual.

**8.1 PARTE I**

Mejorar las características del micrófono:

* Investiga como incrementar el valor de pausa del threshold para dar más tiempo al usuario antes de que el sistema asuma que ha terminado de hablar.
* Investiga como establecer un umbral de energía para filtrar ruidos de fondo.
* Investiga como adaptar el recognizer de ruido ambiental antes de comenzar a escuchar.

**8.2 PARTE II**

Añadir una interfaz gráfica:

* Importaremos tkinter, así como threading, que nos permitirá crear “hilos” en nuestro programa. Un hilo es una unidad de ejecución muy pequeña que puede ejecutarse en paralelo a nuestro proceso principal.

import threading

import time

import tkinter as tk

from tkinter import scrolledtext

* Añade los elementos que compondrán la interfaz gráfica:
  + Ventana:

ventana = tk.Tk()

ventana.title("Asistente Virtual")

ventana.iconbitmap('UD3\_Prueba\_1/icono.png')

ventana.geometry("500x400")

* + Texto:

texto\_conversacion = scrolledtext.ScrolledText(ventana, wrap=tk.WORD)

texto\_conversacion.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

* + Botón:

boton\_hablar = tk.Button(ventana, text="Hablar", command=iniciar\_pedir\_cosas)

boton\_hablar.pack(pady=10)

* + Mainloop:

ventana.mainloop()

Además, tendremos que ejecutar la función principal: pedir\_cosas() mediante threading para que no pise al bucle de la ventana:

def iniciar\_pedir\_cosas():

    threading.Thread(target=pedir\_cosas).start()

Por último, añadiremos las siguientes líneas a la función “hablar” y “transformar\_audio\_a\_texto” para que incorporen su contenido en la interfaz:

* Hablar:

    # Ejecutar y esperar a que termine

    engine.runAndWait()

    texto\_conversacion.insert(tk.END, "Asistente: " + mensaje + "\n")

    texto\_conversacion.see(tk.END)

* Audio\_a\_texto:

            # Devolver el texto

            texto\_conversacion.insert(tk.END, "Usuario: " + texto + "\n")

            texto\_conversacion.see(tk.END)

            return texto

**9. ANEXO: FORMATO DE LOS DOCUMENTOS JSON**

Para que los scripts sean funcionales, tendrás que crear diferentes ficheros .json con el formato adecuado.

Añade a la carpeta raíz los siguientes documentos JSON y complétalos con información relacionada con el proyecto, así como respuestas genéricas para hacer al asistente más interactivo.

**1. equipos.json**

Este archivo contiene información sobre los equipos participantes y sus integrantes.

{

"equipos": [

{

"nombre": "Tigres FC",

"participantes": ["Carlos López", "María Pérez", "Juan Gómez", "Laura Martínez"]

},

{

"nombre": "Leones FC",

"participantes": ["Ana Rodríguez", "Luis Sánchez", "Pedro Díaz", "Elena Fernández"]

}

]

}

**2. partidos.json**

Este archivo contiene información sobre los partidos programados, incluyendo los equipos, fecha, hora y lugar.

{

"partidos": [

{

"equipo1": "Tigres FC",

"equipo2": "Leones FC",

"fecha": "2023-11-25",

"hora": "18:00",

"lugar": "Polideportivo Municipal"

},

{

"equipo1": "Águilas FC",

"equipo2": "Cóndores FC",

"fecha": "2023-11-26",

"hora": "20:00",

"lugar": "Estadio Central"

}

]

}

**3. reglas.json**

Este archivo contiene las reglas del torneo de fútbol sala.

{

"reglas": [

{

"titulo": "Cambios ilimitados",

"descripcion": "Los equipos pueden realizar cambios de jugadores sin límite durante el partido."

},

{

"titulo": "Duración del partido",

"descripcion": "Cada partido tendrá una duración de 40 minutos, divididos en dos tiempos de 20 minutos."

}

]

}

**4. estiramientos.json**

Este archivo proporciona recomendaciones de estiramientos y ejercicios para los jugadores.

{

"ejercicios": [

{

"nombre": "Estiramiento de cuádriceps",

"descripcion": "De pie, sujeta el pie derecho con la mano derecha y tira suavemente hacia los glúteos. Mantén durante 30 segundos y cambia de pierna."

},

{

"nombre": "Estiramiento de isquiotibiales",

"descripcion": "Sentado en el suelo con las piernas extendidas, inclínate hacia adelante y alcanza los dedos de los pies. Mantén durante 30 segundos."

}

]

}

**5. datos.json**

Este archivo contiene patrones y respuestas generales que el asistente puede utilizar para interacciones adicionales.

{

"intents": [

{

"tag": "saludo",

"patterns": ["hola", "buenos días", "buenas tardes", "buenas noches"],

"responses": ["¡Hola! ¿En qué puedo ayudarte?", "Saludos, ¿cómo puedo asistirte?"]

},

{

"tag": "despedida",

"patterns": ["adiós", "hasta luego", "nos vemos", "gracias"],

"responses": ["¡Hasta pronto!", "Fue un placer ayudarte.", "¡Que tengas un buen día!"]

},

{

"tag": "agradecimiento",

"patterns": ["gracias", "muchas gracias", "te lo agradezco"],

"responses": ["No hay de qué.", "Para eso estoy.", "Siempre a tu disposición."]

},

{

"tag": "chiste",

"patterns": ["cuéntame un chiste", "dime un chiste", "quiero reírme"],

"responses": ["¿Por qué el futbolista llevaba una escalera al partido? Porque quería subir al marcador.", "¿Qué le dijo un balón de fútbol a otro? ¡Vamos a rodar un rato!"]

}

]

}

**Notas importantes:**

* **Formato JSON:** Asegúrate de que cada archivo JSON esté correctamente formateado y que las comillas sean dobles ("), ya que el formato JSON lo requiere.
* **Codificación UTF-8:** Guarda todos los archivos con codificación UTF-8 para evitar problemas con caracteres especiales como tildes y eñes.
* **Ubicación de los archivos:** Todos los archivos deben estar en el mismo directorio que tu script de Python, o ajustar las rutas en el código si están en ubicaciones diferentes.
* **Actualización de datos:** Puedes ampliar estos ejemplos agregando más equipos, partidos, reglas, ejercicios y acciones solidarias para enriquecer la interacción con el asistente.

**Asegúrate de que los nombres de los archivos en tu código coincidan con los nombres de los archivos que has creado. Por ejemplo:**

**Código:**

with open('equipos.json', 'r', encoding='utf-8') as archivo\_equipos:

equipos = json.load(archivo\_equipos)

Ahora, con estos archivos JSON correctamente configurados, tu asistente virtual estará listo para funcionar y responder a las consultas basadas en estos datos.