UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SABIA Project's Documentation Innovation Banking Hack Fest 2025

Mario Sanjuan, Santiago Riveros, Emilly de Michelle, Ivan Stoykov, Mariam El Klai, Naoual Khachane

18-10-2025

Contents

1 Introduction

En la actualidad, la digitalización de los servicios financieros ha mejorado la forma en que las personas gestionan su dinero, pero todavía existen grandes desafíos en materia de accesibilidad y usabilidad. Según la Organización Mundial de la Salud, **más de 285 millones de personas en el mundo** viven con algún tipo de discapacidad visual, y en España, solo el 60 por ciento de los adultos mayores utiliza internet de forma regular. Además, los estudios de experiencia de usuario en banca móvil revelan que un nuevo usuario tarda entre **5 y 10 minutos en realizar su primera operación básica**, y entre el 30 y 40 por ciento comete errores durante el proceso.

Con estos datos como punto de partida, nace Sab-ia: tu asistente del día a día, una herramienta diseñada para hacer que gestionar el dinero sea tan fácil como hablar. Nuestra propuesta busca ofrecer una forma segura, rápida y natural de realizar operaciones bancarias, garantizando accesibilidad, inclusión y confianza para todo tipo de usuarios, especialmente aquellos con mayores dificultades para interactuar con interfaces digitales tradicionales.

2 Temáticas elegidas

La aplicación desarrollada aborda tres temáticas principales que se alinean con los objetivos del *Innovation Hackathon Sabadell Fest*: la inclusión digital, el uso de la inteligencia artificial y la ciberseguridad aplicada a transacciones financieras.

2.1 Diseño inclusivo y accesible para colectivos vulnerables

Uno de los principales desafíos consistió en diseñar una experiencia de usuario verdaderamente inclusiva. Más allá de adaptar una aplicación bancaria ya existente, el objetivo fue *replantear por completo la forma de interacción*, priorizando el uso de la voz y la simplicidad en cada paso. De este modo, se garantiza que personas con discapacidad visual o con baja alfabetización digital puedan utilizar la aplicación de manera autónoma, segura y eficiente.

2.2 Uso de la inteligencia artificial

El proyecto incorpora un flujo de **inteligencia artificial multimodal** capaz de procesar las interacciones del usuario de manera natural. Esta IA analiza y clasifica los mensajes recibidos para determinar la intención del usuario, ya sea realizar una consulta o navegar entre las distintas secciones de la aplicación. Además, se integraron modelos de IA para la *conversión de voz a texto* y *de texto a voz*, permitiendo una comunicación fluida y accesible en ambos sentidos.

2.3 Ciberseguridad y transacciones de pago

La seguridad del usuario constituye un pilar esencial del proyecto. La aplicación implementa medidas de autenticación y confirmación de operaciones que garantizan la protección de datos personales y financieros. Asimismo, se contemplan futuras mejoras orientadas a reforzar la **prevención del fraude** y la **protección frente al robo de identidad**, asegurando un entorno de uso confiable y seguro.

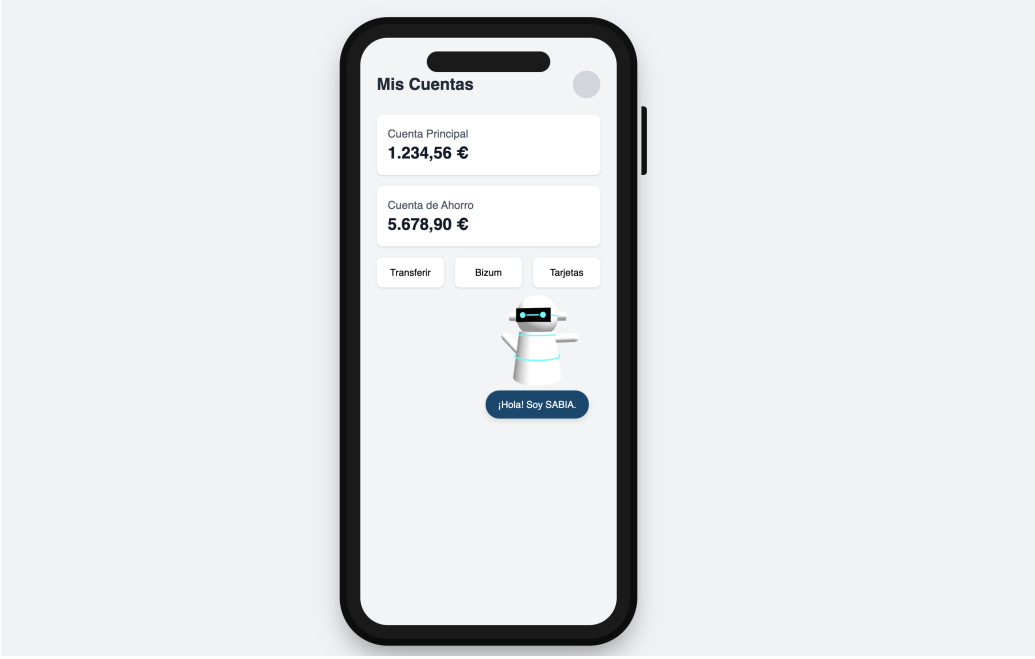
3 Estructura del proyecto

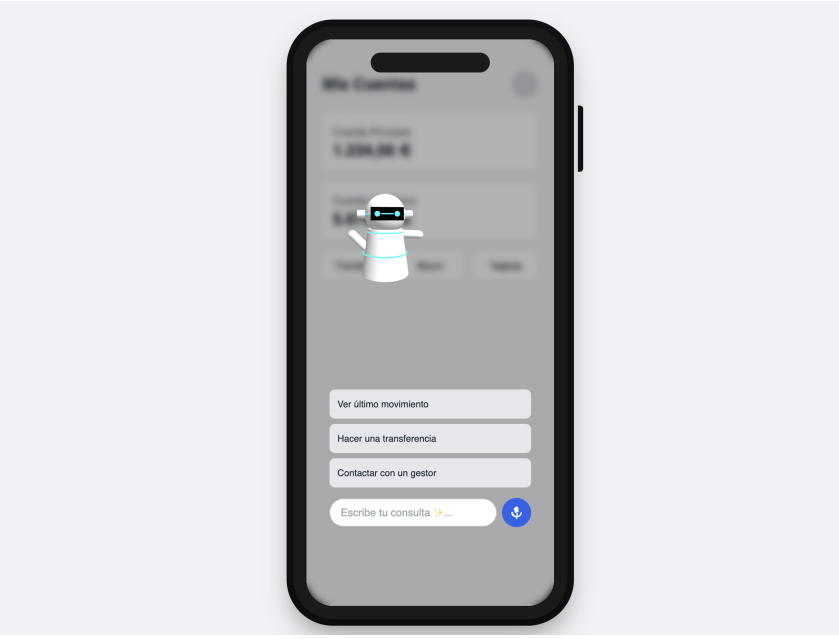
La estructura del proyecto se compone de tres elementos principales: la interfaz web, el backend (que integra la API junto con los modelos de conversión de texto a voz y de voz a texto) y el flujo de inteligencia artificial encargado de procesar las interacciones del usuario

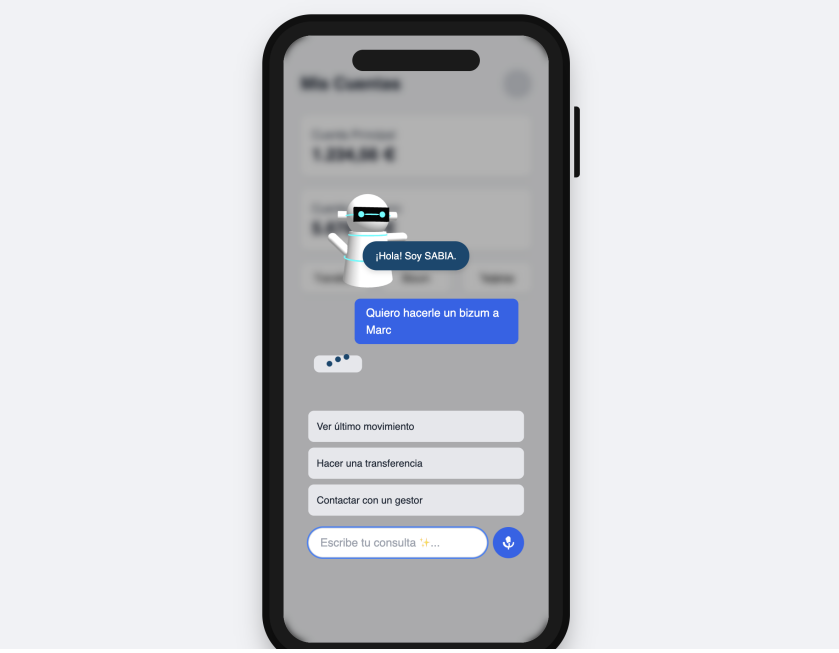
3.1 Frontend - Interfaz web

Para el desarrollo de la interfaz web se utilizaron los lenguajes **html**, **css**, **javascript**, **xml** y **kotlin**. Esta interfaz reproduce la pantalla principal de la aplicación del Banco Sabadell, mostrando el saldo disponible y las diferentes funciones habituales. No obstante, incorporamos nuestra propuesta innovadora: SAB-IA, un asistente virtual en forma de pequeño robot al que el usuario puede dirigirse mediante comandos de voz o a través de un chat integrado.

El asistente es capaz de *guiar al usuario por la aplicación*, ya sea llevándolo directamente a la sección deseada o indicándole cómo acceder a ella. Además, puede **detectar posibles fraudes** que afecten al cliente y **responder consultas básicas** usando información actual de la app del banco sabadell, proporcionando una experiencia más accesible, intuitiva y segura.




Ver último movimiento
Hacer una transferencia
Contactar con un gestor
Escribe tu consulta ✨...


¡Hola! Soy SABIA.
Quiero hacerle un bizum a Marc
Ver último movimiento
Hacer una transferencia
Contactar con un gestor
Escribe tu consulta ✨...

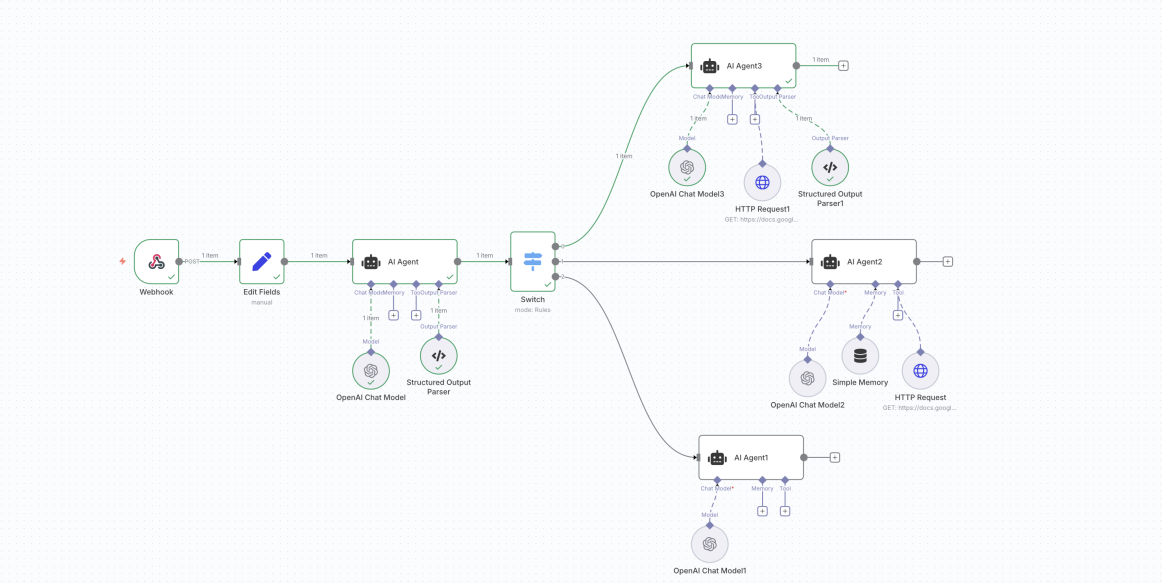
3.2 Flujo de inteligencia artificial

El flujo de inteligencia artificial desarrollado en **n8n** actúa como el núcleo lógico del asistente **SAB-IA**, encargándose de procesar, clasificar y actuar respecto a los mensajes del usuario. El sistema comienza con un *webhook* que recibe las solicitudes enviadas desde la API del backend. Esta información es almacenada y procesada mediante un nodo de edición de campos que normaliza la entrada.

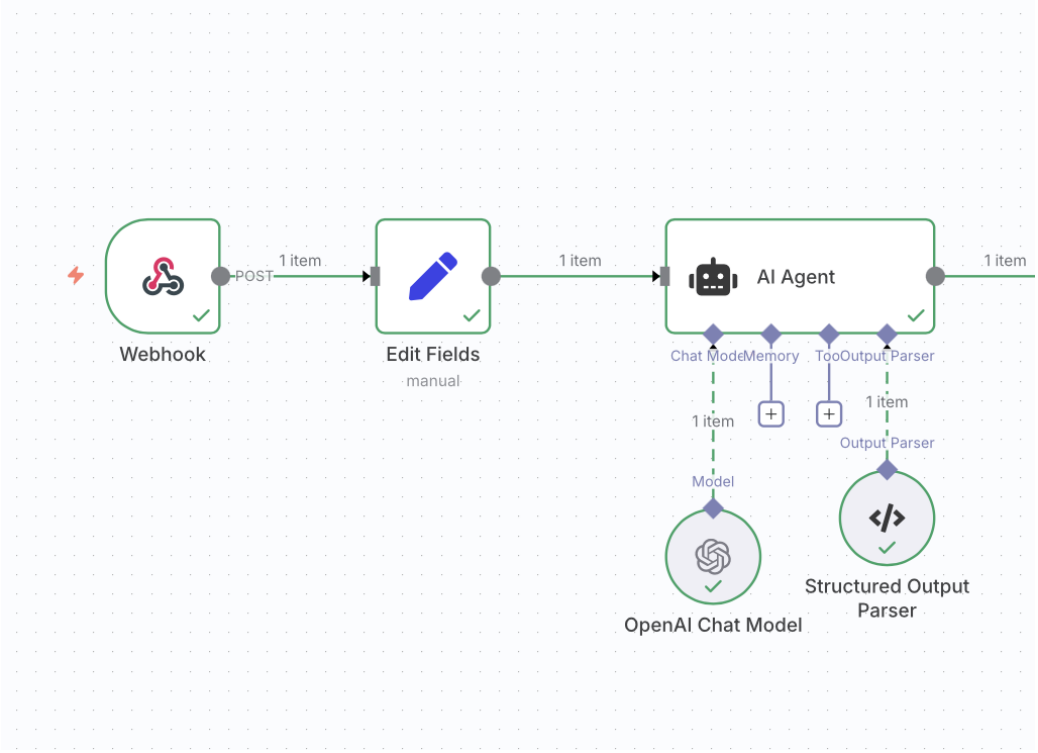
Posteriormente, un agente inteligente se encarga de **clasificar el mensaje en tres categorías**: *Message*, *Fraud* o *Action*. Para ello, el modelo evalúa el contexto del texto recibido y determina si se trata de una consulta general, una acción dentro de la aplicación (como hacer un pago o navegar a otra sección) o una posible alerta de fraude. Esta clasificación se gestiona mediante un nodo *Switch* que deriva cada tipo de mensaje hacia un flujo independiente, permitiendo ejecutar la respuesta más adecuada según el caso.

En función de la categoría identificada, el flujo activa distintos agentes especializados: uno genera respuestas informativas a consultas del usuario, otro emite **alertas preventivas** ante posibles fraudes, y un tercero determina la **navegación interna** dentro de la aplicación según los mapeos definidos en documentos externos de *Google Docs*.

En conjunto, este flujo dota a **SAB-IA** de una inteligencia adaptativa y segura, capaz de **entender, decidir y actuar de forma autónoma** en distintos escenarios de interacción con el cliente.



(a) Diseño del flujo de AI en n8n cloud



(a) Inicio del flujo y clasificador de AI

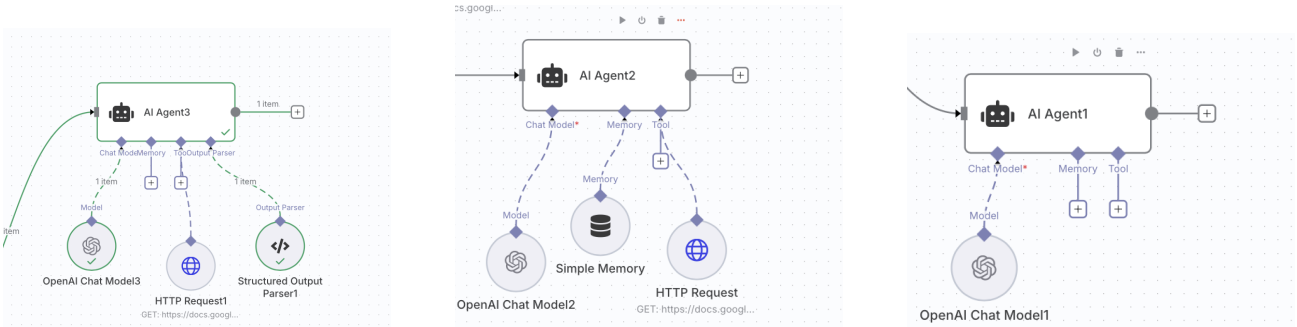
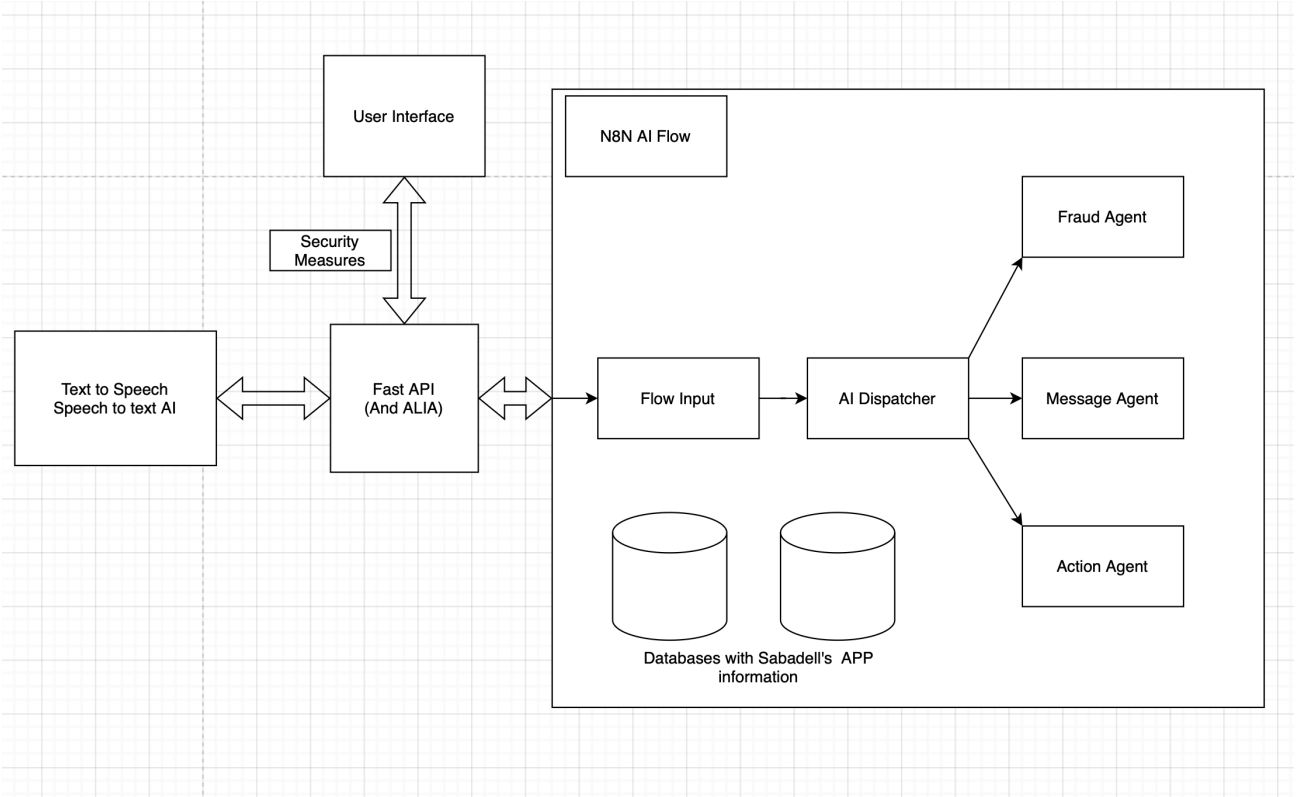


Figure 6: Agentes de AI para cada caso: Acción, Consulta y Fraude

3.3 Backend



(a) Diagrama de la aplicación

En la imagen se muestra la estructura general de la aplicación, con especial énfasis en el funcionamiento del **backend**. Este componente está compuesto por una **FastAPI**, junto con la implementación de los **modelos de inteligencia artificial (ALIA)** y los módulos de **conversión de texto a voz (Text-to-Speech)** y **voz a texto (Speech-to-Text)**.

El backend actúa como la **capa lógica central** del sistema y como un **puente de comunicación** entre la interfaz web y el flujo de inteligencia artificial, gestionando el intercambio de datos y la ejecución de las operaciones solicitadas por el usuario.

A continuación, se presenta una descripción más detallada de los modelos TTS, STT y ALIA, junto con su función dentro del sistema.

3.3.1 Implementación del modelo STT

El backend está compuesto por tres módulos principales. En primer lugar, el archivo `stt.py` implementa la conversión de voz a texto utilizando la API de Gemini "gemini-2.5-flash", permitiendo transformar las entradas de voz del usuario en texto interpretable por el sistema.

El módulo `server.py` gestiona la comunicación entre el backend y la interfaz, encargándose de subir y descargar archivos una vez procesados, ya sean de audio o texto, para que el asistente SAB-IA pueda interactuar correctamente con el usuario.

Por último, `main.py` actúa como el fichero orquestador, responsable de ejecutar y coordinar las diferentes clases y funciones del sistema, asegurando la correcta conexión entre todos los componentes del backend.

3.3.2 Implementación de ALIA

Para mejorar la accesibilidad lingüística de nuestra aplicación, integramos el modelo ALIA – Aitana-TA-2B-S, presentado en el HackFest. Este modelo, alojado en Hugging Face, se conecta al backend mediante una API pública, utilizando el resultado del n8n connector como entrada.

La función translate() actúa como un wrapper que envía texto en español al modelo de traducción español–valenciano, recibe la respuesta en tiempo real (streaming) y devuelve una traducción limpia y lista para su uso.

Se emplearon hiperparámetros equilibrados para obtener traducciones naturales y precisas, evitando literalismos y manteniendo un estilo claro y fluido.

En resumen, hemos usado ALIA para que los hablantes de Valenciano y Euskera también tengan la posibilidad de interactuar también con la aplicación (SabIA)

4 Implementación del código

En esta sección del informe se presentan diversos fragmentos de código (code snippets) que consideramos de especial relevancia para la comprensión y el funcionamiento del proyecto.

4.1 Frontend - User Interface

El siguiente fragmento de código muestra una función implementada para enviar el input del usuario desde la interfaz web hacia el backend, específicamente a través de la API.

```
async function callFastAPI(inputText = "test input") {
    try {
        console.log('Llamando a la API de FastAPI...');
        console.log('URL:', 'http://localhost:8000/predict');
        console.log('Payload:', { input: inputText });
        const response = await fetch('http://localhost:8000/predict',
            {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json', },
            body: JSON.stringify({
                input: inputText }) });
        console.log('Response status:', response.status);
        console.log('Response ok:', response.ok);
        if (response.ok) {
            const data = await response.json();
            console.log('Respuesta de la API:', data);
            return data;
        } else {
            console.error('Error en la respuesta:', response.status);
            const errorText = await response.text();
            console.error('Error text:', errorText);
            throw new Error(`HTTP error! status: ${response.status},
                text: ${errorText}`);
        }
    } catch (error) {
        console.error('Error al llamar a la API:', error);
        console.error('Error type:', error.name);
        console.error('Error message:', error.message);
        // Verificar si es un error de conexión
        if (error.name === 'TypeError' && error.message.includes('
            Failed to fetch')) {
            throw new Error('No se puede conectar con la API.
                Verifica que esté corriendo en http://127.0.0.1:8000')
                ;
        } throw error;}}
```

4.2 Fast API Text Input Endpoint Code

```python

@app.post("/predict")
def predict(data: InputData):
    #Received input from the frontend
    input_text = data.input

    print(f"API WORKS - Received input: {input_text}")

    #Send text to the ai multimodel and wait for the response
    n8n_response = n8n.request_models(input_text)

    # Parsear la respuesta JSON a diccionario
    try:
        import json
        if isinstance(n8n_response, str):
            n8n_dict = json.loads(n8n_response)
        else:
            n8n_dict = n8n_response

        print(f"N8N Response parsed: {n8n_dict}")

    except (json.JSONDecodeError, TypeError) as e:
        print(f"Error parsing N8N response: {e}")
        n8n_dict = {"error": "Failed to parse N8N response", "raw_response":
           str(n8n_response)}
    #Ver que caso es cada una
    if "path" in n8n_dict:
        return {
        "path": n8n_dict["path"],
        "Instructions for the users": n8n_dict["Instructions for the users"]
        }
    else:
        # Extraer el contenido del output si existe
        if "output" in n8n_dict and "output" in n8n_dict["output"]:
            return {
                "message": n8n_dict["output"]["output"],
                "raw_response": n8n_dict
            }
        elif "output" in n8n_dict:
            return {
                "message": n8n_dict["output"],
                "raw_response": n8n_dict }
        else:
            return {
                "message": str(n8n_dict),
                "raw_response": n8n_dict}
```

4.3 STT Model Code

Falta por unir el código de los modelos TTS y STT con la api y hacer la implementación del input de audio del proyecto.

```python
import os
from google import genai
from google.genai.errors import APIError

GOOGLE_API_KEY = "**************************************************"

try:
    client = genai.Client(api_key=GOOGLE_API_KEY)
except Exception as e:
    print(f"Error al inicializar el cliente de Gemini.")
    print(f"Detalles del error: {e}")
    exit()

FILE_PATH = r'C:\Users\santi\PycharmProjects\deep\output_audio.wav'

TRANSCRIPTION_PROMPT = 'Genera una transcripción del discurso en este audio.'

MODEL_NAME = 'gemini-2.5-flash'

if not os.path.exists(FILE_PATH):
    print(f"Error: El archivo no se encontró en la ruta especificada: {
        FILE_PATH}")
    print("Asegúrate de reemplazar 'path/to/your/audio.mp3' con la ruta
        correcta a tu archivo.")
else:
    myfile = None
    try:
        myfile = client.files.upload(file=FILE_PATH)

        response = client.models.generate_content(
            model=MODEL_NAME,
            contents=[TRANSCRIPTION_PROMPT, myfile]
        )
        print(response.text)
    except APIError as e:
        print(f"\nError de la API de Gemini: {e}")
        print("Verifica que tu clave de API sea correcta y que el archivo de
            audio sea compatible.")
    except Exception as e:
        print(f"\nOcurrió un error inesperado: {e}")
    finally:
        if myfile:
            client.files.delete(name=myfile.name)
```

4.4 TTS Model Code

Este es el código lo que hace es convertir un input de texto a voz (un audio)

```python
import wave
import contextlib

from google import genai
# Get API key from environment variable
GOOGLE_API_KEY = "**************************************************"

if not GOOGLE_API_KEY:
    print("Error: GOOGLE_API_KEY environment variable not set.")
else:
    client = genai.Client(api_key=GOOGLE_API_KEY)
    MODEL_ID = "gemini-2.5-flash-preview-tts"
    @contextlib.contextmanager
    def wave_file(filename, channels=1, rate=24000, sample_width=2):
        with wave.open(filename, "wb") as wf:
            wf.setnchannels(channels)
            wf.setsampwidth(sample_width)
            wf.setframerate(rate)
            yield wf

    response = client.models.generate_content(
      model=MODEL_ID,
      contents="Di 'Quieres que realice la tranferencia usando tus datos' en
          español de España",
      config={"response_modalities": ['Audio']},
    )

    blob = response.candidates[0].content.parts[0].inline_data
    print(blob.mime_type)

    # Save the audio to a file
    fname = 'output_audio.wav'
    with wave_file(fname) as wav:
        wav.writeframes(blob.data)

    print(f"Audio saved to {fname}")
    # You would then open 'output_audio.wav' with a media player to listen.
```

4.5 Implementación de traducción para accesibilidad lingüística con ALIA

```
--- main.py ---
import server, stt, os, n8n_connector, ALIA


FILE_PATH_TO_UPLOAD = r"C:\Users\santi\PycharmProjects\deep\PoliHackers\
    Backend\Santiago\output_audio.wav"

FILENAME_TO_DOWNLOAD = os.path.basename(FILE_PATH_TO_UPLOAD)
DOWNLOAD_SAVE_PATH = r"C:\Users\santi\PycharmProjects\deep\PoliHackers\
    Backend\Santiago\download_audio.wav"


def main():
    server.upload_audio(FILE_PATH_TO_UPLOAD)
    server.download_file(FILENAME_TO_DOWNLOAD, DOWNLOAD_SAVE_PATH)
    response = stt.sptotext(DOWNLOAD_SAVE_PATH)
    translation = ALIA.translate(response)
    n8n_connector.request_models(response)
    print(translation)


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

```
--- ALIA.py ---

from openai import OpenAI

hf_token = ""


def translate(input: str) -> str:
    client = OpenAI(
        base_url="https://y9qb4ck1bebf371m.us-east-1.aws.endpoints.
            huggingface.cloud/v1/",
        api_key=hf_token
    )
    chat_completion = client.chat.completions.create(
        model="gplsi/Aitana-TA-2B-S",
        messages=[
            {
                "role": "user",
                "content": f"Translate the following text from Spanish into
                    Valencian.\nSpanish: {input} \nValencian:"
            }
        ],
```

```
        stream=True,
        temperature=0.5,
        max_tokens=400,
        frequency_penalty=1.2
    )
    traduccion = ""
    for message in chat_completion:
        delta = message.choices[0].delta.content
        if delta:
            traduccion += delta

    return traduccion.strip()
```