

Translator AI.

Manuel Parrado Barrero

PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL I.E.S. Portada Alta

Índice

1.	Arquitectura y flujo de trabajo general.....	2
1.2.	Funcionalidad	2
1.3.	Arquitectura y flujo de trabajo	2
2.	Módulos de la arquitectura	4
2.1.	Comunicación	4
2.2.	Servicios Web.....	5
2.2.1.	/url/api/upload_record_to_s3 (POST).....	5
2.2.2.	/url/api/get_all_record_from_S3 (GET).....	6
2.2.3.	/url/api/transcribe_selected_record (GET)	6
2.2.4.	/url/api/translate_text_from_transcribe (POST)	7
2.2.5.	/url/api/translate_text_from_transcribe (POST)	7
2.3.	Tareas	8
2.3.1.	Convertir un audio a texto.....	8
2.3.2.	Traducción del texto transcrito	10
2.3.3.	Generación de audio traducido.....	12
2.4.	Nodos de almacenamiento	13
3.	Ejemplo de uso	15
4.	Recursos necesarios para ejecutar la aplicación.....	16

1. Arquitectura y flujo de trabajo general

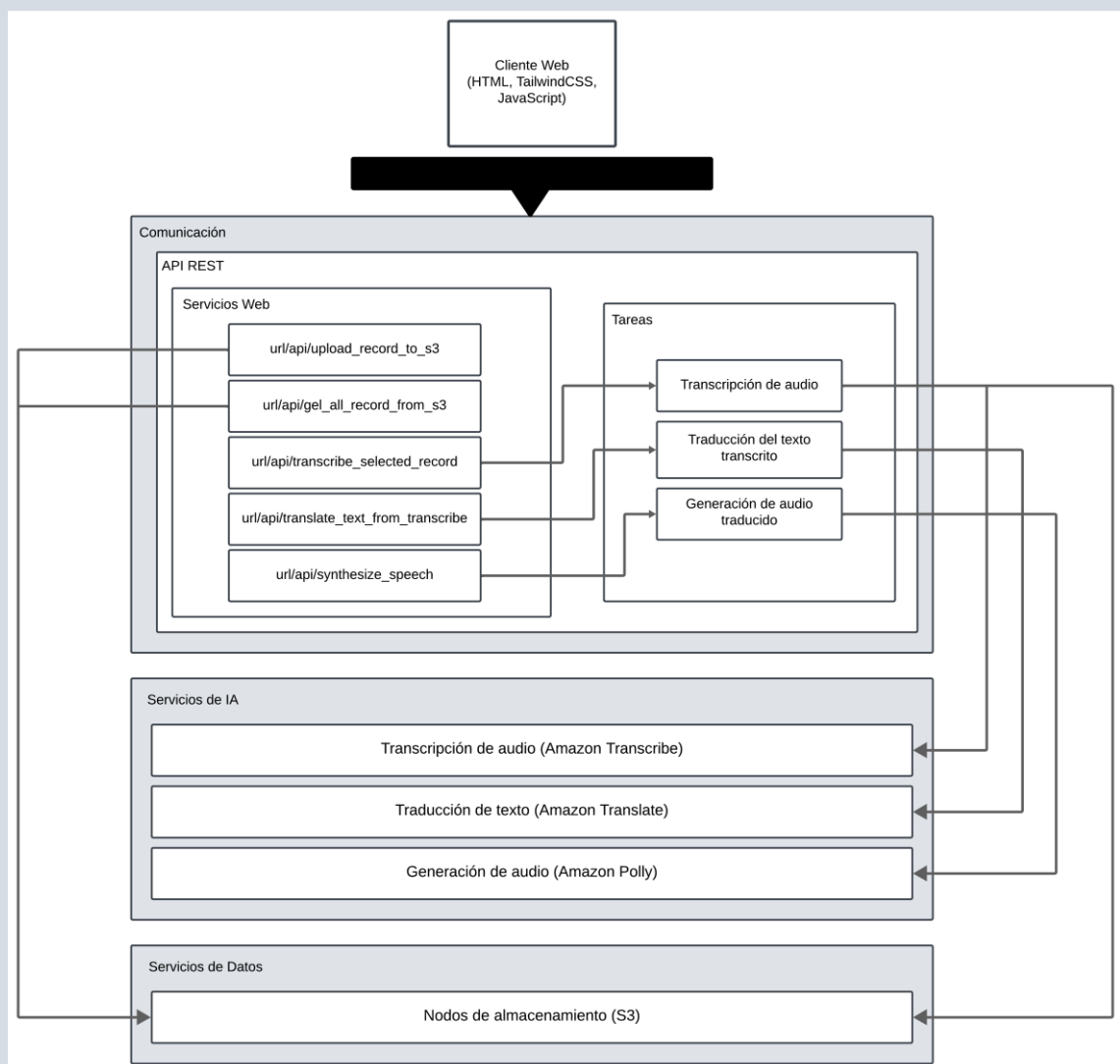
Para empezar con la documentación, vamos a ver brevemente la arquitectura y la funcionalidad del proyecto de AWS “Translator AI”.

1.2. Funcionalidad

La principal funcionalidad de la aplicación es subir grabaciones de audio y poder traducirlas a otros idiomas. Para esto hago uso de diferentes tecnologías y servicios de IA de Amazon Web Service.

1.3. Arquitectura y flujo de trabajo

Después de haber explicado por encima la funcionalidad, vamos a ver la arquitecta de la aplicación.



Visión general de la arquitectura de la aplicación

En la imagen podemos ver un esquema general de que vamos a explicar más profundidad a continuación:

1. El usuario, desde la interfaz web, sube su grabación al nodo de almacenamiento.
2. La aplicación hace una petición al nodo de almacenamiento, para recoger todos los nombres de la grabaciones alojadas.
3. El usuario elige mediante un desplegable el nombre de la grabación que quiere procesar. Este debe indicar el idioma en que se habla en la grabación, para que el resultado de la transcripción sea correcto. Cuando haya seleccionado estos dos parámetros, pulsará el botón “Iniciar transcripción”.
4. Cuando se haya pulsado y validado que los datos de entrada son correctos, se iniciará un proceso que indicará al servicio Amazon Transcribe que archive del nodo de almacenamiento deberá transcribir y en que idioma está. Este hará la transcripción (la página se queda en espera hasta que termine el proceso) y la mandará al cliente para mostrarla en pantalla.
5. Al haber terminado la transcripción, aparecerá en pantalla el formulario de traducción, en el que indicaremos mediante un desplegable, el idioma al que queremos traducir. Al pulsar el botón “Traducir”, hará una petición al servicio Amazon Translate para traducir el texto y acto seguido pasará el texto traducido a Amazon Polly, que devolverá un audio con el texto traducido. Cuando todas estas peticiones hayan terminado, se mostrará por pantalla el texto traducido, un botón de “play” para reproducir el audio y un botón de descarga que permitirá descargar dicho audio.

2. Módulos de la arquitectura

2.1. Comunicación

Explicado de manera breve, la comunicación de mi aplicación esta de hecha de forma que un archivo de código JavaScript mediante promesas, hace peticiones a la API en Python. Este para darle la información y hacer los procesos que necesita el cliente, llama a los servicios elegidos de AWS.

Función ejemplo de petición a la API en Python desde JavaScript:

```
// Función para cargar grabaciones en el dropdown
Codeium: Refactor | Explain | X
async function loadRecordings() {
  try {
    const response = await fetch('http://127.0.0.1:5000/api/get_all_record_from_s3', {
      method: 'GET',
    });

    if (!response.ok) {
      throw new Error("Error al obtener las grabaciones");
    }

    const recordings = await response.json();

    recordings.forEach(recording => {
      const option = document.createElement('option');
      option.value = recording.name;
      option.textContent = recording.name;
      recordingsDropdown.appendChild(option);
    });

    console.log("Grabaciones cargadas correctamente:", recordings);
  } catch (error) {
    console.error("Error al cargar las grabaciones:", error);
    showAlert('error', "No hay grabaciones disponibles.");
  }
}
```

Esta función pide, de manera asíncrona, los nombres de las grabaciones alojadas en el nodo de almacenamiento y las muestra en el desplegable.

Función ejemplo de petición al servicio de AWS de la API en Python:

```
# Endpoint para obtener los nombres de las grabaciones, para mostrarlas en el desplegable
Codeium: Refactor | Explain | Generate Docstring | X
@app.application.route('/api/get_all_record_from_s3', methods=['GET'])
def get_all_record_from_s3():
    try:
        # Listar todos los objetos (archivos) del bucket
        print(f"Listando todos los archivos en el bucket '{bucket_audio}'...")
        bucket = s3.Bucket(bucket_audio)
        files = [{"name": obj.key} for obj in bucket.objects.all()]

        # Verificar si se encontraron archivos
        if not files:
            print("No se encontraron archivos en el bucket.")
            return Response("No se encontraron archivos en el bucket.", status=404)

        print(f"Archivos encontrados: {files}")
        return jsonify(files) # Devuelve una lista en formato JSON
    except Exception as e:
        print(f"Error al obtener los archivos del bucket: {str(e)}")
        return Response(f"Error al obtener los archivos del bucket: {str(e)}", status=500)
```

Esta función llama al servicio S3, para obtener todos archivos de audio guardados en el nodo de almacenamiento (el nombre del bucket lo obtiene del archivo “.env”) y mandarlos al JavaScript si todo ha ido correctamente o mostrarle un error.

2.2. Servicios Web

En la aplicación utilizo 5 servicios web, que ofrecen la distinta información y procesos que necesita la aplicación para funcionar.

2.2.1. /url/api/upload_record_to_s3 (POST)

Este servicio permite subir archivos de audio al nodo de almacenamiento.

```
# Endpoint para subir archivos a S3
Codeium: Refactor | Explain | Generate Docstring | X
@app.application.route('/api/upload_record_to_s3', methods=['POST'])
def upload_record_to_s3():
    try:
        # Obtén el archivo del request
        file = request.files.get('file')

        # Valida que se haya recibido un archivo
        if not file:
            print("No se encontró un archivo en la solicitud.")
            return Response("No se encontró un archivo en la solicitud", status=400)

        # Sube el archivo a S3
        print(f"Subiendo el archivo '{file.filename}' a S3...")
        s3.Bucket(bucket_audio).put_object(Key=file.filename, Body=file)
        print(f"Archivo '{file.filename}' subido exitosamente a S3.")

        # Responde con éxito
        return Response(f"Archivo '{file.filename}' subido correctamente a S3.", status=200)
    except Exception as e:
        print(f"Error al subir el archivo: {str(e)}")
        return Response(f"Error al subir el archivo: {str(e)}", status=500)
```

2.2.2. /url/api/get_all_record_from_S3 (GET)

Este servicio hace una llamada al servicio de S3, para obtener los nombres de todas las grabaciones almacenadas en el nodo de almacenamiento.

```
# Endpoint para obtener los nombres de las grabaciones, para mostrarlas en el desplegable
Codeium: Refactor | Explain | Generate Docstring | X
@app.application.route('/api/get_all_record_from_s3', methods=['GET'])
def get_all_record_from_s3():
    try:
        # Listar todos los objetos (archivos) del bucket
        print(f"Listando todos los archivos en el bucket '{bucket_audio}'...")
        bucket = s3.Bucket(bucket_audio)
        files = [{"name": obj.key} for obj in bucket.objects.all()]

        # Verificar si se encontraron archivos
        if not files:
            print("No se encontraron archivos en el bucket.")
            return Response("No se encontraron archivos en el bucket.", status=404)

        print(f"Archivos encontrados: {files}")
        return jsonify(files) # Devuelve una lista en formato JSON
    except Exception as e:
        print(f"Error al obtener los archivos del bucket: {str(e)}")
        return Response(f"Error al obtener los archivos del bucket: {str(e)}", status=500)
```

2.2.3. /url/api/transcribe_selected_record (GET)

Este servicio recoge del cliente el nombre de la grabación, el nombre formateado para que no tenga problemas con el servicio de AWS y el lenguaje de la grabación, y le pasa estos datos a la función que hace la petición al servicio de AWS, que transcribe el audio. Esta función le devolverá el texto y se lo pasará al cliente.

```
# Endpoint para llamar a la función de transcripción
Codeium: Refactor | Explain | Generate Docstring | X
@app.application.route('/api/transcribe_selected_record', methods=['GET'])
def transcribe_selected_record_endpoint():
    try:
        real_name = request.args.get('realName')
        filename = request.args.get('filename')
        language = request.args.get('language')

        # Validación
        if not filename:
            return Response("No se proporcionó un nombre de archivo.", status=400)

        # Llamar a la función de transcripción
        transcribed_text = transcribe_audio(real_name, filename, language)

        # Devolver el texto transcrito
        return jsonify({'transcription': transcribed_text})
    except Exception as e:
        print(f"Error en la transcripción: {str(e)}")
        return Response(f"Error en la transcripción: {str(e)}", status=500)
```

2.2.4. /url/api/translate_text_from_transcribe (POST)

Este servicio recoge el texto transcrito y el lenguaje indicado por el cliente. Llama a la función que hace la petición al servicio de Amazon Translate y cuando la función le devuelve el texto traducido, se lo pasa al cliente.

```
# Endpoint para hacer la traducción del texto sacado del audio
Codeium: Refactor | Explain | Generate Docstring | X
@app.route('/api/translate_text_from_transcribe', methods=['POST'])
def translate_text_endpoint():
    try:
        data = request.json
        text = data.get('text')
        target_language = data.get('language')

        # Validación
        if not text or not target_language:
            return Response("Faltan datos: 'text' y/o 'language'.", status=400)

        # Llamar a la función de traducción
        translated_text = translate_text(text, target_language)

        # Responder con el texto traducido
        return jsonify({'translated_text': translated_text})

    except Exception as e:
        print(f"Error en la traducción: {str(e)}")
        return Response(f"Error en la traducción: {str(e)}", status=500)
```

2.2.5. /url/api/translate_text_from_transcribe (POST)

Este servicio recibe el texto traducido y el código de lenguaje en el que debe generar el audio. Llama a la función y esta hace la petición al servicio de Amazon Polly, que genera el audio y le devuelve el audio en el idioma deseado, después se lo pasa al cliente.

```
# Endpoint para Llamar a La función de generación de audios
Codeium: Refactor | Explain | Generate Docstring | X
@app.route('/api/synthesize_speech', methods=['POST'])
def synthesize_speech_function():
    try:
        data = request.json
        text = data.get('text')
        language_code = data.get('language')

        # Validación
        if not text or not language_code:
            return Response("Faltan datos: 'text' y/o 'language'.", status=400)

        # Llamar a la función para generar el audio
        audio_stream = generate_audio(text, language_code)

        # Devolver el audio como respuesta binaria
        return Response(
            audio_stream,
            content_type='audio/mp3',
            headers={"Content-Disposition": "attachment; filename=audio.mp3"}
        )

    except Exception as e:
        print(f"Error en el endpoint de síntesis de voz: {e}")
        return Response(f"Error en el endpoint de síntesis de voz: {e}", status=500)
```


2.3. Tareas

En la aplicación hago 3 tareas principales, estas tareas son transcripción de audio (convertir un audio en texto), traducción del texto transcrito (traducir el texto convertido del audio al idioma que el cliente indique) y generación de audio traducido (generamos un audio a partir del texto traducido).

2.3.1. Convertir un audio a texto

Se cargan las variables de entorno necesarias y se inicia una sesión en el servicio Transcribe.

```
# Cargamos las variables de entorno
load_dotenv()

# Recuperamos las credenciales y configuraciones
access_key_id = os.environ.get('ACCESS_KEY_ID')
secret_access_key = os.environ.get('ACCESS_SECRET_KEY')
bucket_audio = os.environ.get('BUCKET_AUDIO')

# Inicializamos el cliente de Transcribe
transcribe = boto3.Session(
    aws_access_key_id=access_key_id,
    aws_secret_access_key=secret_access_key,
    region_name='eu-west-2'
).client('transcribe')
```

Aquí muestro la función que se encarga de llamar al servicio Amazon Transcribe, que se encarga de transcribir y devolver el código a la API, que se devolverá al cliente.

```
# Función para transcribir un archivo de audio de S3
Codeium: Refactor | Explain | Generate Docstring | X
def transcribe_audio(real_name, filename, language):
    try:
        # Crear un nombre único para el trabajo de transcripción
        job_name = f"transcribe-{filename.replace('.', '-')}-{int(time.time())}"

        # Iniciar el trabajo de transcripción
        response = transcribe.start_transcription_job(
            TranscriptionJobName=job_name,
            Media={'MediaFileUri': f's3://{bucket_audio}/{real_name}'},
            MediaFormat='mp3',
            LanguageCode=f'{language}'
        )
        print(f"Trabajo de transcripción '{job_name}' iniciado: {response}")

        # Monitorear el estado del trabajo
        while True:
            job_response = transcribe.get_transcription_job(TranscriptionJobName=job_name)
            status = job_response['TranscriptionJob']['TranscriptionJobStatus']

            if status in ['COMPLETED', 'FAILED']:
                break
            time.sleep(5)
```

Resto de la función

```
# Monitorear el estado del trabajo
while True:
    job_response = transcribe.get_transcription_job(TranscriptionJobName=job_name)
    status = job_response['TranscriptionJob']['TranscriptionJobStatus']

    if status in ['COMPLETED', 'FAILED']:
        break
    time.sleep(5)

# Procesar el resultado
if status == 'COMPLETED':
    transcript_uri = job_response['TranscriptionJob']['Transcript']['TranscriptFileUri']
    response = requests.get(transcript_uri)
    transcription = response.json()
    transcribed_text = transcription['results']['transcripts'][0]['transcript']
    return transcribed_text

raise Exception(f"Error en la transcripción: {job_response['TranscriptionJob']['FailureReason']}")

except Exception as e:
    print(f"Error en la transcripción: {str(e)}")
    raise
```

Llamadas al servicio

Primero hacemos la petición que iniciar el trabajo de transcripción. En esta llamada le debemos pasar el nombre que va a tener la tarea (TranscriptionJobName), el nombre del audio y donde esta alojado (Media), el formato del archivo (MediaFormat) y el lenguaje del audio (LanguageCode).

```
# Iniciar el trabajo de transcripción
response = transcribe.start_transcription_job(
    TranscriptionJobName=job_name,
    Media={'MediaFileUri': f's3://{bucket_audio}/{real_name}'},
    MediaFormat='mp3',
    LanguageCode=f'{language}'
)
```

Respuesta

```
{
  "TranscriptionJob": {
    "TranscriptionJobName": "transcribe-espa-ol_1-mp3-1733907549",
    "TranscriptionJobStatus": "IN_PROGRESS",
    "LanguageCode": "es-ES",
    "MediaFormat": "mp3",
    "Media": {
      "MediaFileUri": "s3://manuel-parrado-proyectoaws-audios/español_1.mp3"
    },
    "StartTime": "2024-12-11T09:59:10.267000+00:00",
    "CreationTime": "2024-12-11T09:59:10.251000+00:00"
  },
  "ResponseMetadata": {
    "RequestId": "8393580d-e7bd-4a20-a2a7-ac1c59f05381",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "8393580d-e7bd-4a20-a2a7-ac1c59f05381",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "307",
      "date": "Wed, 11 Dec 2024 08:59:09 GMT"
    },
    "RetryAttempts": 0
  }
}
```

La segunda llamada se encarga de monitorear la tarea en curso, para indicar al cliente si la tarea esta en progreso, completada o ha fallado. Aquí debemos pasarle el nombre que la tarea que queremos monitorear.

```
# Monitorear el estado del trabajo
while True:
    job_response = transcribe.get_transcription_job(TranscriptionJobName=job_name)
    status = job_response['TranscriptionJob']['TranscriptionJobStatus']

    if status in ['COMPLETED', 'FAILED']:
        break
    time.sleep(5)
```

Respuesta

```
{
  "TranscriptionJob": {
    "TranscriptionJobName": "transcribe-espa-ol_1-mp3-1733908083",
    "TranscriptionJobStatus": "COMPLETED",
    "LanguageCode": "es-ES",
    "MediaSampleRateHertz": 48000,
    "MediaFormat": "mp4",
    "Media": {
      "MediaFileUri": "s3://manuel-parrado-proyectoaws-audios/espa\u00f1ol_1.mp3"
    },
    "Transcript": {
      "TranscriptFileUri": "https://s3.eu-west-2.amazonaws.com/aws-transcribe-eu-west-2-prod/56"
    },
    "StartTime": "2024-12-11T10:08:04.844000+00:00",
    "CreationTime": "2024-12-11T10:08:04.826000+00:00",
    "CompletionTime": "2024-12-11T10:08:46.196000+00:00",
    "Settings": {
      "ChannelIdentification": false,
      "ShowAlternatives": false
    }
  },
  "ResponseMetadata": {
    "RequestId": "f4dfbe94-0951-45ec-a8db-513b7fa8d38d",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "f4dfbe94-0951-45ec-a8db-513b7fa8d38d",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "2403",
      "date": "Wed, 11 Dec 2024 09:08:50 GMT"
    },
    "RetryAttempts": 0
  }
}
```

2.3.2. Traducción del texto transcrito

Cargamos las variables de entorno necesario e iniciamos una sesión del servicio Translate.

```
# Cargar las variables de entorno
load_dotenv()

# Configuración de AWS
access_key_id = os.environ.get('ACCESS_KEY_ID')
secret_access_key = os.environ.get('ACCESS_SECRET_KEY')

# Inicializar el cliente de Translate
translate_client = boto3.Session(
    aws_access_key_id=access_key_id,
    aws_secret_access_key=secret_access_key,
    region_name='eu-west-2'
).client('translate')
```

Aquí muestro la función que se encarga llamar al servicio de Amazon Translate, que se encarga de traducir el texto que se ha transcrito, este nuevo texto traducido se enviará al cliente mediante la API.

```
# Traduce el texto sacado del audio al idioma que se le indique
Codeium: Refactor | Explain | Generate Docstring | X
def translate_text(text, target_language):
    try:
        print(text, "Archivo de traducción")
        print(target_language, "Archivo de traducción")

        response = translate_client.translate_text(
            Text=text,
            SourceLanguageCode='auto', # Detecta automáticamente el idioma de origen
            TargetLanguageCode=target_language
        )

        print(response['TranslatedText'], "Archivo de traducción")

        return response['TranslatedText']
    except Exception as e:
        print(f"Error en la traducción: {str(e)}")
        raise
```

Llamada al servicio

En esta llamada debemos pasarle como parámetros el texto que queremos traducir (Text), el idioma del texto lo indica que lo detecte automáticamente (SourceLanguageCode) y le indicamos el idioma al que queremos traducir el texto (TargetLanguageCode). Esta petición devolverá el texto traducido.

```
response = translate_client.translate_text(
    Text=text,
    SourceLanguageCode='auto', # Detecta automáticamente el idioma de origen
    TargetLanguageCode=target_language
)
```

Respuesta

```
{
  "TranslatedText": "Hello. My name is Manuel.",
  "SourceLanguageCode": "es",
  "TargetLanguageCode": "en",
  "ResponseMetadata": {
    "RequestId": "fbbcae91-d5b1-4446-b8e1-45c3a1baa0a2",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "fbbcae91-d5b1-4446-b8e1-45c3a1baa0a2",
      "cache-control": "no-cache",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "98",
      "date": "Wed, 11 Dec 2024 09:15:59 GMT"
    },
    "RetryAttempts": 0
  }
}
```

2.3.3. Generación de audio traducido

Cargamos las variables de entorno necesarias e iniciamos el servicio de generación de audio Polly.

```
# Cargar variables de entorno
load_dotenv()

# Configuración de AWS
access_key_id = os.environ.get('ACCESS_KEY_ID')
secret_access_key = os.environ.get('ACCESS_SECRET_KEY')

# Crear cliente de Amazon Polly
polly_client = boto3.Session(
    aws_access_key_id=access_key_id,
    aws_secret_access_key=secret_access_key,
    region_name='eu-west-2' # Cambia según tu región
).client('polly')
```

Aquí podemos ver la función que utilizar el servicio de Amazon Polly, para generar un audio con el texto traducido para poder reproducirlo y descargarlo en el cliente. Este audio se mandará a la API y después al cliente.

```
# Función para generar un audio con Polly
Codeium: Refactor | Explain | Generate Docstring | X
def generate_audio(text, language_code):
    try:
        # Configurar voz según el idioma
        voice_id = "Joanna" # Voz predeterminada
        voices = {
            "es": "Lucia",      # Español (España)
            "es-MX": "Mia",     # Español (Latinoamérica)
            "en": "Joanna",     # Inglés (General)
            "en-GB": "Amy",     # Inglés (Británico)
            "fr": "Celine",     # Francés
            "de": "Vicki",      # Alemán
            "it": "Carla",      # Italiano
            "pt": "Camila",     # Portugués (Brasil)
            "zh": "Zhiyu",      # Chino Simplificado
            "ja": "Mizuki"      # Japonés
        }

        # Configurar voz basada en el código del idioma
        voice_id = voices.get(language_code, voice_id)
```

Resto de la función.

```
# Generar el audio con Polly
response = polly_client.synthesize_speech(
    Text=text,
    OutputFormat="mp3",
    VoiceId=voice_id
)

# Retornar el flujo de audio
return response['AudioStream'].read()
except Exception as e:
    print(f"Error al generar audio con Polly: {e}")
    raise
```

Llamada al servicio

Para llamar al servicio de generación de audio debemos pasarle como parámetros el texto traducido que queremos reproducir (Text), el formato que queremos para el audio que genere (OutputFormat) y el nombre de la voz que queremos usar para nuestro audio (VoiceId). Dependiendo del idioma el nombre de la voz será diferente, para cuidar acentos. Esto devolverá el audio generado.

```
# Generar el audio con Polly
response = polly_client.synthesize_speech(
    Text=text,
    OutputFormat="mp3",
    VoiceId=voice_id
)
```

Respuesta

```
{
  "TranslatedText": "Hello. My name is Manuel.",
  "SourceLanguageCode": "es",
  "TargetLanguageCode": "en",
  "ResponseMetadata": {
    "RequestId": "de9190a9-a6f6-4324-86d3-950898fbb7da",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "de9190a9-a6f6-4324-86d3-950898fbb7da",
      "x-amzn-requestcharacters": "25",
      "content-type": "audio/mpeg",
      "transfer-encoding": "chunked",
      "date": "Wed, 11 Dec 2024 09:18:19 GMT"
    },
    "RetryAttempts": 0
  },
  "ContentType": "audio/mpeg",
  "RequestCharacters": 25,
  "AudioStream": "<StreamingBody object>"
}
```

2.4. Nodos de almacenamiento

Para almacenar las grabaciones que el usuario sube, utilizamos un bucket del servicio de S3.

The screenshot shows the Amazon S3 console interface. On the left is a navigation menu with options like 'Buckets de uso general', 'Buckets de directorio', 'Buckets de tablas', 'Concesiones de acceso', 'Puntos de acceso', 'Puntos de acceso del objeto', 'Lambda', 'Puntos de acceso de varias regiones', 'Operaciones por lotes', 'Analizador de acceso de IAM para S3', and 'Configuración de bloqueo de acceso público correspondiente a esta cuenta'. The main panel displays the details for a bucket named 'manuel-parrado-proyectoaws-audios'. It includes tabs for 'Objetos', 'Propiedades', 'Permisos', 'Métricas', 'Administración', and 'Puntos de acceso'. Under the 'Objetos (4)' tab, there are buttons for 'Copiar URI de S3', 'Copiar URL', 'Descargar', 'Abrir', 'Eliminar', 'Acciones', 'Crear carpeta', and 'Cargar'. Below these buttons is a table listing the objects:

Nombre	Tipo	Última modificación	Tamaño	Clase de almacenamiento
aleman.mp3	mp3	10 Dec 2024 1:14:03 PM CET	73.0 KB	Estándar
español_1.mp3	mp3	9 Dec 2024 9:40:20 PM CET	88.9 KB	Estándar
español_2.mp3	mp3	9 Dec 2024 9:40:31 PM CET	127.9 KB	Estándar
frances.mp3	mp3	9 Dec 2024 9:42:53 PM CET	13.2 KB	Estándar

Peticiones al servicio S3

Petición para subir la grabación al bucket.

```
# Sube el archivo a S3
print(f"Subiendo el archivo '{file.filename}' a S3...")
s3.Bucket(bucket_audio).put_object(Key=file.filename, Body=file)
print(f"Archivo '{file.filename}' subido exitosamente a S3.")
```

Petición para recoger los nombres de todos las grabaciones almacenadas en el bucket.

```
# Listar todos los objetos (archivos) del bucket
print(f"Listando todos los archivos en el bucket '{bucket_audio}'...")
bucket = s3.Bucket(bucket_audio)
files = [{"name": obj.key} for obj in bucket.objects.all()]
```

3. Ejemplo de uso

Primero subimos el archivo de audio, para poder procesarlo.



Translator AI.

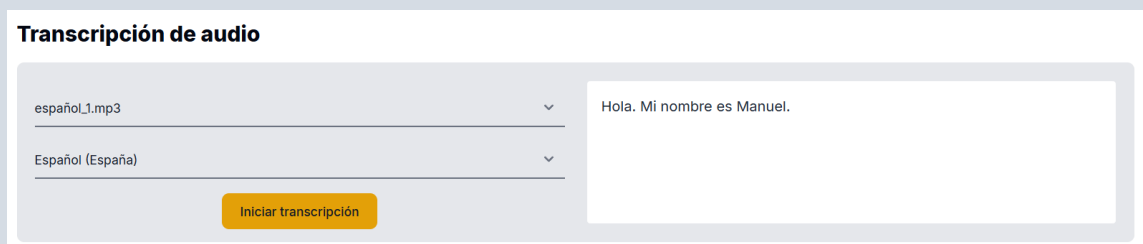
Pequeño proyecto con AWS, para la asignatura de Programación de Inteligencia Artificial. traducirlas a otros idiomas. Para esto utilizo los servicios de S3, Amazon Transcribe, Ama

Archivo subido correctamente. **Subir grabación para poder traducirla**

Subir grabación

Seleccionar archivo **Subir**

Después, con la grabación ya subida, en el desplegable la seleccionamos, seleccionamos el idioma de la grabación y pulsamos el botón de “Iniciar transcripción”. Mientras las transcripción se esté haciendo, se mostrará a la derecha “Procesando la transcripción ...”. Cuando se haya terminado, mostrará el texto transcrito del audio y el formulario para traducir y generar el audio.

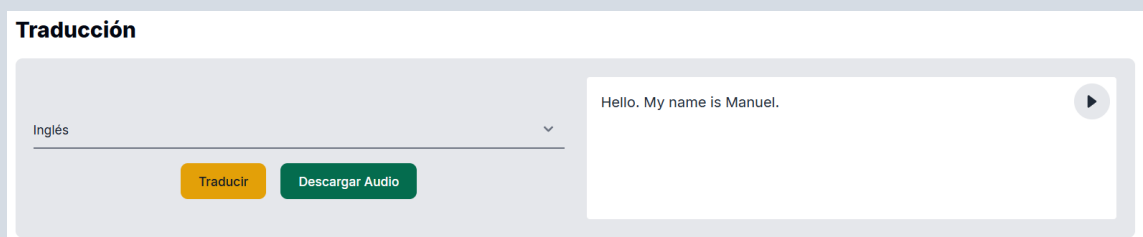


Transcripción de audio

Iniciar transcripción


Hola. Mi nombre es Manuel.

El formulario de traducción, aparecerá cuando se haya terminado la transcripción. En este debemos seleccionar el idioma de traducción y pulsaremos el botón “Traducir”. Mientras se traduce y genera el audio, se mostrará a la derecha “Traduciendo...”. Cuando se haya terminado el proceso aparecerá el texto traducido, un botón de “play” para reproducir el audio generado y un botón de descarga para descargarlo.



Traducción

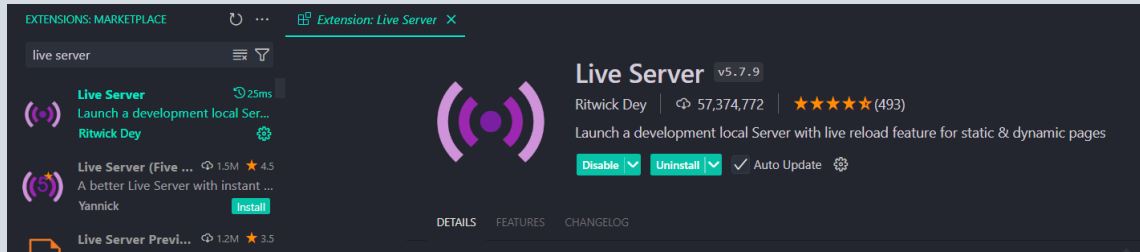
Traducir **Descargar Audio**

Hello. My name is Manuel. 

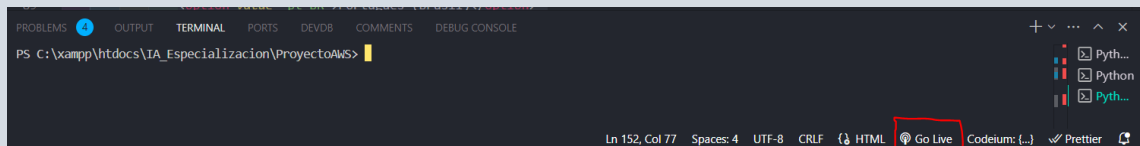
4. Recursos necesarios para ejecutar la aplicación

Live Server

Para ejecutar en local se necesita la extensión de Visual Studio Code llamada “Live Server”.



Cuando se haya instalado nos debemos dirigir al archivo HTML “main_view.html”, y abajo en la barra de Visual Studio Code, deberemos clicar en el icono que pone “Go Live”. Esto habilitará un servidor web, para poder probar correctamente el HTML.



Crear entorno virtual e instalar librerías necesarias

Abrimos el proyecto y creamos el entorno virtual.

```
PS D:\ProyectoAWS> python -m venv venv
PS D:\ProyectoAWS>
```

Activamos el entorno virtual.

```
PS D:\ProyectoAWS> venv\Scripts\activate
>>
(venv) PS D:\ProyectoAWS>
```

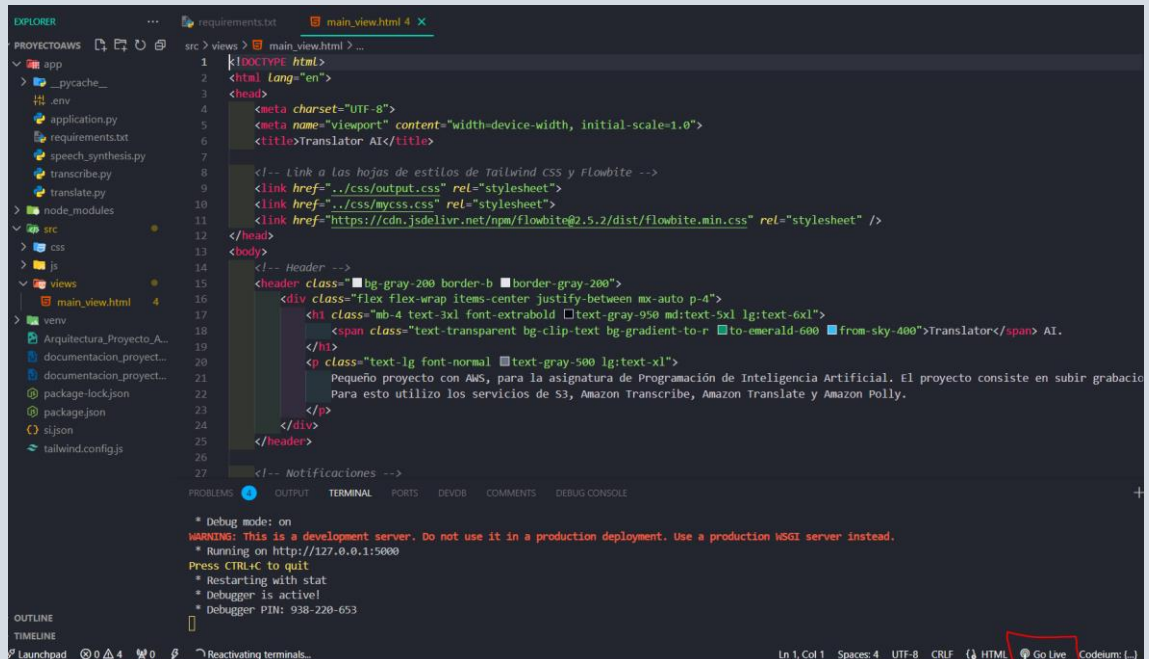
Entramos a la app y ejecutamos el comando para instalar las librerías necesarias para que la aplicación funcione.

```
(venv) PS D:\ProyectoAWS> cd .\app\
(venv) PS D:\ProyectoAWS\app> pip install -r .\requirements.txt
```

Cuando se hayan instalado las librerías, ejecutamos el archivo application.py.

```
(venv) PS D:\ProyectoAWS\app> python application.py
>>
* Serving Flask app 'application'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 938-220-653
```

Ahora en Visual Studio Code, abrimos el archivo “main_view.html” y clicamos el botón de la extensión “Live Server”.



Al iniciar el server y la pagina por primera vez es posible que las grabaciones no se carguen en el desplegable. Si pasa esto basta con recargar la página para que aparezcan las grabaciones subidas al nodo de almacenamiento.