Codigo Incorporación APIs

Ivan Stoykov

October 2025

1 Introducción

En el proyecto se busca implementar un sistema de reconocimiento de voz y procesamiento de texto basado en Flask y Python. El objetivo es diseñar una forma de poder communicar todas las API a través de un router de manera segura, escalable y sostenible.

2 Arquitectura

La arquitectura se puede separar en tres partes:

apy.py (Router): Actúa como punto central de comunicación entre la interfaz y los servicios internos. Dependiendo del tipo de información recibida (audio o texto), decide si debe enviarla a API A (en el caso de audio) o directamente a API B (en el caso de texto). Además, gestiona el modo de transmisión —local o remoto— y se encarga de reenrutar al destino el texto transcrito que proviene de API A.

speech-to-text.py (STT/API A): Implementa un servicio de conversión de voz a texto basado en Gemini, encargado de procesar los archivos de audio recibidos y generar su transcripción. Posteriormente, los resultados pueden enviarse al Router o directamente a API B (según se defina en futuras integraciones, como la incorporación de ALIA).

api_b.py (API B): Funciona como un endpoint de prueba que permite verificar la correcta recepción y transmisión de datos a través del Router y de API A. En versiones posteriores será reemplazada por la API definitiva que formará parte del sistema principal.

3 Configuración

La configuración funciona mediante variables de entorno, para permitir una fácil portabilidad entre entornos de desarrollo y producción.

Algunas variables clave incluyen:

- `API_A_MODE`: determina si el ASR se ejecuta como proceso local o vía HTTP.

- `API_A_SCRIPT`: ruta al script local de transcripción.
- `API_B_URL`: destino de los resultados procesados.
- `ROUTER_HMAC_SECRET`: clave usada para firmar los mensajes del webhook.
- `ASR_MODEL`: nombre del modelo utilizado por Gemini.

Para poder ejecutar el sistema completo en local:
1) Lanzar API B (python api_b.py)
2) Lanzar el router export API_B_URL=http://127.0.0.1:7000/intent
3) Lanzar el router (python api.py)
4) (Opcional) Ejecutar el ASR local python speech-to-text.py –file audio.wav

4 APIs

Router(api.py): Expone tres endpoints principales:

POST/ingest: recibe datos en formato de texto o audio. Si es la entrada es text lo envia directamente a API B. Si la entrada es audio, se reenviar el archivo a la API A remota o local según el modo de configuración.

POST /from-api-a: es el endpoint de webhook que permite a la API A o al script local de transcripción devolver el texto procesado al router. Este endpoint implementa un sistema de verificación HMAC mediante la cabecera X-Signature, garantizando la integridad y autenticidad de los datos recibidos.

GET /healthz: proporciona información básica sobre el estado de la aplicación y la validez de la configuración actual, permitiendo comprobar si el servicio está operativo.

API B (api_b.py): La API B actúa como un punto de prueba para verificar la comunicación entre los diferentes componentes del sistema. Recibe texto proveniente del router y devuelve una respuesta sencilla confirmando la recepción y el procesamiento correcto de los datos. En futuras versiones, este servicio se sustituirá por la API definitiva que gestionará el flujo real de información dentro del proyecto.

API A (speech-to-text.py):

El componente API A, también denominado servicio de Speech-to-Text (STT), se encarga de transformar archivos de audio en texto legible utilizando el modelo Gemini.

Su función principal es recibir un archivo de audio —ya sea directamente desde el router o a través de un proceso local—, procesarlo con el modelo seleccionado y generar la transcripción correspondiente. Una vez obtenida la transcripción, el servicio puede actuar de dos maneras, según la configuración del entorno:

Modo local (process): el script se ejecuta de forma directa en el mismo entorno que el router. Tras completar la conversión de audio a texto, envía el resultado al router mediante una solicitud HTTP al endpoint de webhook (/from-api-a). En este flujo, se utiliza una clave HMAC compartida para firmar las respuestas y garantizar su autenticidad.

Modo remoto (http): el router delega el procesamiento a un servidor externo configurado como API A. Este servicio remoto recibe el archivo, realiza la transcripción y devuelve el resultado al router o a la API B mediante una llamada de retorno.

Además, el script está diseñado para integrarse con otros sistemas futuros, como ALIA, permitiendo ampliar las funcionalidades de análisis o interpretación semántica del texto. Gracias a su modularidad y al uso de variables de entorno, puede adaptarse fácilmente a distintos modelos de lenguaje, rutas de archivo o puntos de notificación (webhooks), manteniendo la seguridad y el control del flujo de datos.

5 Seguridad

El sistema incorpora distintos mecanismos de autenticación, integridad y protección de datos con el objetivo de garantizar la fiabilidad de las comunicaciones entre los componentes.

Cada webhook que llega al endpoint /from-api-a utiliza un esquema de verificación HMAC (Hash-based Message Authentication Code) basado en el algoritmo SHA-256. Antes de aceptar la información recibida, el router calcula una firma local a partir del cuerpo del mensaje y la compara con la firma enviada en la cabecera X-Signature. De esta forma, se asegura que el contenido no haya sido alterado durante la transmisión y que provenga de una fuente legítima.

Las claves utilizadas para este proceso, como ROUTER_HMAC_SECRET, se gestionan mediante variables de entorno para evitar su exposición dentro del código fuente o del repositorio.

Además, se recomienda desplegar el router detrás de un proxy inverso (por ejemplo, Nginx o Traefik) configurado con HTTPS, con el fin de proteger las comunicaciones frente a interceptaciones y garantizar la confidencialidad de los datos transmitidos.

En conjunto, estas medidas proporcionan una capa de seguridad sólida que abarca tanto la autenticación de los mensajes como la protección de las credenciales y la integridad del flujo de información.