Arquitectura Final: VTV en un Sistema RAG Desacoplado (API + React)

Este documento describe la arquitectura de servicios completa para implementar la funcionalidad **Voz-a-Voz (VTV)** en el sistema de Generación Aumentada por Recuperación (RAG), utilizando Python (FastAPI) como *backend* y React.js como *frontend*.

1. Visión General de la Arquitectura

La arquitectura final está compuesta por tres capas independientes, lo que permite la escalabilidad y la modernización continua de cada componente.

Capa	Componentes Principales	Responsabilidad	Estrategia RAG Clave
Frontend (Cliente)	React.js, JavaScript (Web Speech API)	Experiencia del usuario, captura de voz (ASR) y reproducción de audio (TTS).	Interfaz VTV
Backend (Servicio RAG)	Python (FastAPI), LangChain	Ejecución de la lógica RAG, búsqueda en la base de datos y generación de texto.	**Chunking Semántico**
Infraestructura	ChromaDB, Ollama/Mistral, Gemini TTS API	Almacenamiento vectorial, Modelo de Lenguaje Grande (LLM) y Servicio de Síntesis de Voz.	Base de datos vectorial

2. Flujo de Interacción Voz-a-Voz (VTV)

El proceso conversacional de principio a fin consta de 6 pasos distribuidos entre el cliente React y el servicio FastAPI.

Secuencia Detallada

Paso	Actor	Acción	Descripción Técnica
1. Captura de Voz	Usuario	El usuario habla después de presionar el botón del micrófono en React.	Se activa el **Web Speech API** de JavaScript en el navegador.
2. Transcripción (ASR)	React.js	El navegador convierte la señal de audio en texto (el *prompt*).	La librería SpeechRecognition de JavaScript entrega la cadena de texto final.
3. Petición al Backend	React.js	Envía el texto transcrito (el *prompt*) al servicio RAG.	Petición POST /api/v1/ask con el cuerpo JSON: { "query": "el texto del usuario" }.
4. Ejecución RAG	FastAPI (Python)	Procesa la *query* con la cadena RAG.	**La fase de *Retrieval* se basa en documentos indexados previamente con Chunking Semántico.** La cadena RAG busca en ChromaDB y llama al LLM (Ollama) para generar la respuesta_textual.
5. Respuesta al Frontend	FastAPI (Python)	Devuelve la respuesta_textual a React.	Respuesta JSON: { "answer": "La respuesta generada.", "sources": [...] }.

6. Síntesis y Reproducción (TTS)	React.js	Llama a la API de Gemini TTS y reproduce el audio.	React llama a gemini-2.5-flash-preview-tts con la respuesta_textual, recibe el audio base64, lo convierte a un objeto Blob y lo reproduce con el elemento <audio>.

3. Tecnologías de Implementación (Stack)

3.1. Backend (Servicio RAG)

Tecnología	Rol	Justificación
Python	Lenguaje Principal	Entorno ideal para bibliotecas de ML/LLM como LangChain.
FastAPI	API REST Framework	Alto rendimiento, soporte asíncrono nativo (crucial para RAG) y documentación automática (Swagger).
LangChain	Orquestación RAG y **Chunking Semántico**	Se utiliza para el preprocesamiento de documentos, aplicando el particionamiento basado en la coherencia semántica.
ChromaDB	Base de Datos Vectorial	Almacenamiento local y eficiente de las incrustaciones de documentos.
Ollama	LLM Servidor	Permite ejecutar modelos de código abierto (Mistral,

		Llama 3) localmente con una interfaz RESTful sencilla.

3.2. Frontend (Aplicación VTV)

Tecnología	Rol	Justificación
React.js (o Next.js)	Framework UI	Permite construir una interfaz de chat moderna, reactiva y con manejo de estado complejo.
Tailwind CSS	Estilismo**	Facilita la creación de una interfaz responsiva y estéticamente agradable.
Web Speech API	ASR (Voz a Texto)	Se ejecuta directamente en el cliente, eliminando la necesidad de un servicio ASR propio.
Fetch/Axios	Cliente HTTP	Para comunicarse con el endpoint /api/v1/ask de FastAPI.

4. Mejores Prácticas Finales

A. Estrategia de Chunking Semántico (CRÍTICO)

- **Implementación:** Utilizar algoritmos de **Chunking Semántico** (como los basados en la variación de incrustaciones o modelos de resumen) en la fase de indexación (offline). Esto asegura que cada fragmento (*chunk*) almacenado en ChromaDB sea una unidad de significado completa, mejorando significativamente la precisión del *retriever*.
- **Impacto en el RAG:** Al garantizar que los *chunks* sean contextualmente densos, reducimos el riesgo de que la respuesta del LLM sea inexacta debido a información incompleta o fragmentada.

B. Rendimiento y Latencia (VTV)

- **TTS Asíncrono en Cliente:** La llamada a la API de Gemini TTS debe realizarse directamente desde React para evitar que el servidor Python actúe como un cuello de

botella de latencia.

- **ASR Preciso:** En un entorno real, la Web Speech API debe tener un mecanismo de *fallback* a un servicio en la nube (ej. Google Cloud) si la precisión es insuficiente para el español conversacional.

C. Mantenimiento del Código

- **Separación de Responsabilidades:** El *backend* de FastAPI **solo** debe generar texto y proporcionar fuentes. El *frontend* de React **solo** debe manejar la interacción de audio (ASR/TTS) y la visualización.
- **Control de Errores:** Implementar manejo de errores CORS en FastAPI y usar try...catch en React para las llamadas API fallidas, notificando al usuario con una respuesta de voz como: *"Lo siento, ha ocurrido un error de conexión."*