Documento Técnico: Sistema de Análisis de Conversaciones Médicas (ElSol-Challenge)

A. Análisis del Requerimiento

Funcionalidades Propuestas

Para satisfacer el requerimiento del cliente, se propuso un sistema basado en un pipeline de procesamiento de lenguaje natural con las siguientes funcionalidades clave:

1. **Ingesta y Transcripción de Audio:** Capacidad de recibir archivos de audio y convertirlos a texto de alta fidelidad.
2. **Extracción de Información Estructurada y No Estructurada:** Un módulo de IA que analiza el texto para identificar y clasificar datos clave como nombres, síntomas, fechas y contexto general.
3. **Almacenamiento Vectorial Inteligente:** Una base de datos diseñada para almacenar no solo los datos, sino también su significado semántico, permitiendo búsquedas conceptuales.
4. **Consulta Conversacional (Chatbot):** Una API que permite a los usuarios hacer preguntas en lenguaje natural y recibir respuestas precisas basadas en la información almacenada.

Decisiones Técnicas y Justificación

Componente	Tecnología Elegida	Justificación (Alternativa Gratuita)	Alternativa para Producción
Framework API	**FastAPI**	Alto rendimiento, soporte nativo `async`, validación de datos con Pydantic y auto-documentación interactiva, ideal para un desarrollo rápido y robusto.	Mantener FastAPI o considerar un framework más completo como Django si se añaden muchas más funcionalidades no relacionadas con la API.
Orquestación	**Docker & Docker Compose**	Garantiza un entorno de desarrollo reproducible, simplifica la configuración de dependencias complejas (API + DB) y facilita el despliegue.	**Kubernetes (EKS, GKE, AKS)** para escalabilidad automática, alta disponibilidad y gestión avanzada de contenedores.
Transcripción	**`faster-whisper` (local)**	Ofrece una precisión excelente (comparable a APIs de pago) sin costos por uso y con control total sobre el modelo. Su rendimiento optimizado es ideal para un MVP.	**Google Speech-to-Text** o **Azure Speech Service**, ya que ofrecen funcionalidades avanzadas como la diarización de hablantes (diferenciar paciente de promotor).

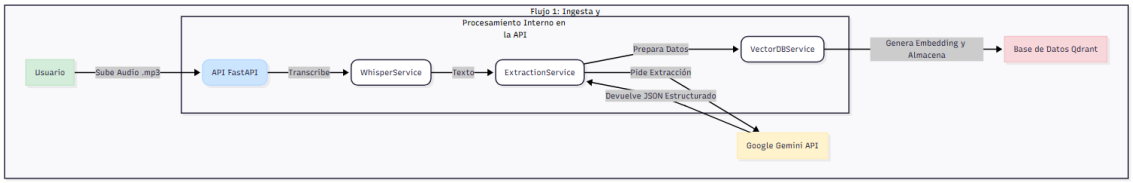
Extracción/Generación	**Google Gemini API (Flash)**	Su modo JSON garantiza una salida estructurada y fiable, crucial para la consistencia de los datos. El modelo Flash es rápido y costo-efectivo.	**Azure OpenAI Service (GPT-4)** por su integración empresarial y políticas de seguridad. O, para casos muy específicos, un **modelo open-source (Llama 3) fine-tuned** con datos médicos anonimizados.
Base de Datos Vectorial	**Qdrant (Docker)**	Es una base de datos vectorial nativa, extremadamente rápida y con un potente motor de filtrado de metadatos. Su imagen de Docker la hace trivial de desplegar.	**Qdrant Cloud** para una solución gestionada, o alternativas como **Pinecone**. Si ya se usa PostgreSQL, **pgvector** puede ser una opción integrada.

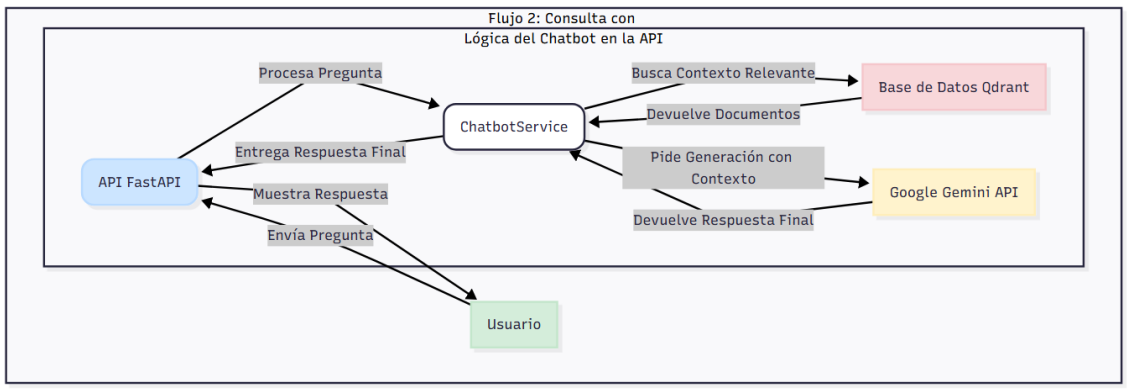
Supuestos Hechos

- **Calidad del Audio:** Se asume que los audios de entrada son de calidad suficiente para una transcripción precisa.
- **Un solo Paciente por Conversación:** El sistema actual está optimizado para extraer información de un solo paciente por audio.
- **Idioma:** Aunque el modelo de transcripción es multilingüe, el prompt de extracción está optimizado para el español.
- **Seguridad:** El MVP no implementa autenticación ni autorización en los endpoints.

B. Arquitectura Propuesta

Diagrama del Sistema





Flujo de Datos

1. **Procesamiento de Audio (`/transcribe`):**
    - El usuario sube un archivo de audio al endpoint de FastAPI.
    - El `WhisperService` transcribe el audio a texto.
    - El `ExtractionService` envía el texto a la API de Gemini con un prompt que solicita una salida en formato JSON estructurado.
    - El `VectorDBService` toma la transcripción, genera un vector de embedding y almacena el vector junto con el JSON extraído (payload) en la colección de Qdrant.
    - La API devuelve un `record_id` confirmando el almacenamiento.
2. **Consulta del Chatbot (`/chat`):**
    - El usuario envía una pregunta en JSON al endpoint.
    - El `ChatbotService` convierte la pregunta en un vector de embedding.
    - Realiza una búsqueda de similitud en Qdrant usando el vector de la pregunta para encontrar los registros más relevantes.
    - Construye un nuevo prompt para Gemini que incluye la pregunta original y los registros recuperados como "contexto".
    - Gemini genera una respuesta en lenguaje natural basada únicamente en el contexto proporcionado.
    - La API devuelve la respuesta al usuario.

C. Plan de Desarrollo

¿Qué se hizo en esta entrega (MVP)?

Se entregó un sistema backend completo y funcional que cumple con todos los requisitos mínimos de la prueba. El sistema es capaz de procesar un archivo de audio desde cero, almacenarlo de forma inteligente y responder preguntas sobre él, todo orquestado en un entorno reproducible con Docker Compose.

Siguientes Pasos (Roadmap)

3. **Subida de Documentos (PDF/Imágenes):**
    - Crear un nuevo endpoint `/upload_document`.
    - Integrar `PyMuPDF` para extraer texto de PDFs y `Tesseract` (vía `pytesseract`) para aplicar OCR a las imágenes.
    - El texto extraído seguiría el mismo pipeline: extracción de entidades con Gemini y almacenamiento en Qdrant.
4. **Diferenciación de Hablantes (Diarization):**

    - Reemplazar `faster-whisper` con una librería más avanzada como `WhisperX`, que puede identificar y etiquetar diferentes hablantes ("SPEAKER_00", "SPEAKER_01").
    - Modificar el prompt de extracción para que entienda el rol de cada hablante (ej. "Extrae los síntomas mencionados por el SPEAKER_00 (paciente)").

5. **Seguridad y Autenticación:**
    - Implementar un sistema de autenticación basado en tokens (ej. OAuth2 con JWT) en FastAPI para proteger todos los endpoints.
    - Añadir un sistema de control de acceso para que los usuarios solo puedan consultar la información de sus pacientes.
6. **Desarrollo de un Frontend:**
    - Crear una interfaz de usuario simple con React o Vue.js que permita subir archivos fácilmente y tener una ventana de chat interactiva.

¿Cómo llevar este sistema a producción?

- **Infraestructura Cloud:** Desplegar los servicios en un orquestador de contenedores como **Google Kubernetes Engine (GKE)** o **AWS EKS**. Esto permitirá escalar los servicios de forma independiente según la carga.
- **Base de Datos Gestionada:** Migrar de un Qdrant en Docker a **Qdrant Cloud** o una instancia de Qdrant autogestionada en Kubernetes con almacenamiento persistente para garantizar la durabilidad y backups.
- **Seguridad Avanzada:**
    - Utilizar un gestor de secretos como **Google Secret Manager** o **AWS Secrets Manager** para las claves de API, en lugar de archivos `.env`.
    - Colocar la API detrás de un **API Gateway** y un **Web Application Firewall (WAF)** para protección contra ataques comunes.
- **MLOps (Machine Learning Operations):**
    - **CI/CD:** Configurar un pipeline en **GitHub Actions** que, ante cada `push` a la rama principal, ejecute tests, construya las imágenes de Docker y las despliegue automáticamente en Kubernetes.
    - **Monitorización y Logging:** Integrar **Prometheus** y **Grafana** para monitorizar el rendimiento de la API (latencia, errores 5xx) y el uso de recursos de los contenedores. Centralizar los logs de la aplicación con **Loki** o un servicio como Datadog.