Public

Documento Técnico — Plataforma RAG Clínico (ElSol)

Jairo Adolfo Céspedes Plata

Los pasos de la ejecución se encuentran en el README.md del repositorio base

Versión: MVP + Extensiones

Cada archivo en el código también está perfectamente documentado si se require más información.

A. Análisis del requerimiento

1) Funcionalidades propuestas

El sistema implementa una plataforma de apoyo a la gestión clínica capaz de integrar y consultar información de pacientes a partir de múltiples orígenes: audio (entrevistas o visitas domiciliarias), documentos PDF e imágenes (formatos escaneados o fotos de reportes). Las funcionalidades clave son:

- Ingesta de archivos (.mp3/.wav, .pdf, .png/.jpg/.jpeg/.webp).
- Transcripción de audio con Whisper (faster-whisper) y extracción de texto de PDFs/imágenes con pdfplumber + EasyOCR.
- Extracción de información estructurada (paciente, edad, fecha) y no estructurada (síntomas, observaciones) con un enfoque híbrido: LLM (Azure OpenAI/OpenAI) con fallback por regex.
- Indexación en un almacén vectorial (Chroma), mediante un embedding local (sentence-transformers) con persistencia en disco, e incorporación de metadata para filtros.
- Endpoint /chat que implementa RAG: recupera chunks relevantes y con un LLM compone la respuesta con citas.
- Cliente en Streamlit para subir archivos y consultar al asistente.
- Seguridad básica con JWT, control de acceso por roles y cifrado de credenciales (hash).

2) Decisiones técnicas y justificación

- **Framework backend:** FastAPI por su alto rendimiento, tipado, documentación automática, y compatibilidad con async.

Public

- **ASR:** faster-whisper local para evitar costos por token/audio y facilitar pruebas en entornos sin internet; en producción se podría alternar con Azure Speech si se requiere escalado gestionado y SLA.
- **OCR:** pdfplumber primero (texto embebido) y EasyOCR como fallback universal (evita dependencias nativas complejas como Tesseract en Docker multi-arquitectura).
- **Vector DB:** Chroma con persistencia en volumen Docker para rapidez de desarrollo; en producción proponemos Qdrant o pgvector por robustez, alta disponibilidad y herramientas de monitoreo.
- **LLM:** integración con Azure OpenAI/OpenAI mediante un contenedor de servicio (app/llm.py) para abstraer proveedor; permite cambiar parámetros de seguridad y temperature.
- **Seguridad:** JWT con passlib (bcrypt) y autorización por roles (promotor, medico, admin); dummy store de usuarios con hash para no almacenar contraseñas en claro.
- **Contenedores:** Dockerfile slim para reducir superficie de ataque y tiempos de build; docker-compose para orquestar backend, frontend y almacenamiento persistente.
- **Frontend:** Streamlit para entrega rápida de un panel funcional de subida y consulta.

3) Supuestos realizados

- Los nombres de pacientes pueden venir tanto en audio como en los documentos; si no aparece, se etiqueta como “desconocido”.
- La fecha se normaliza a YYYY-MM-DD cuando se identifica en el texto; de lo contrario, se usa la fecha actual al indexar.
- La metadata mínima para búsquedas incluye: source_id, patient_name, date y age.
- El LLM no debe inventar datos; se guía con prompts restrictivos y RAG. Aun así, se devuelve contexto y citas.
- El almacenamiento de usuarios es de demostración (dummy); se asume que en producción se integrará un IAM real.
- Se asume que todo lo que entra para el mismo nombre es el mismo usuario.

4) Buenas prácticas

- Dockerización del backend y frontend para ejecución consistente.
- Autenticación JWT con control de roles (promotor, medico, admin).
- Cifrado de contraseñas con bcrypt.

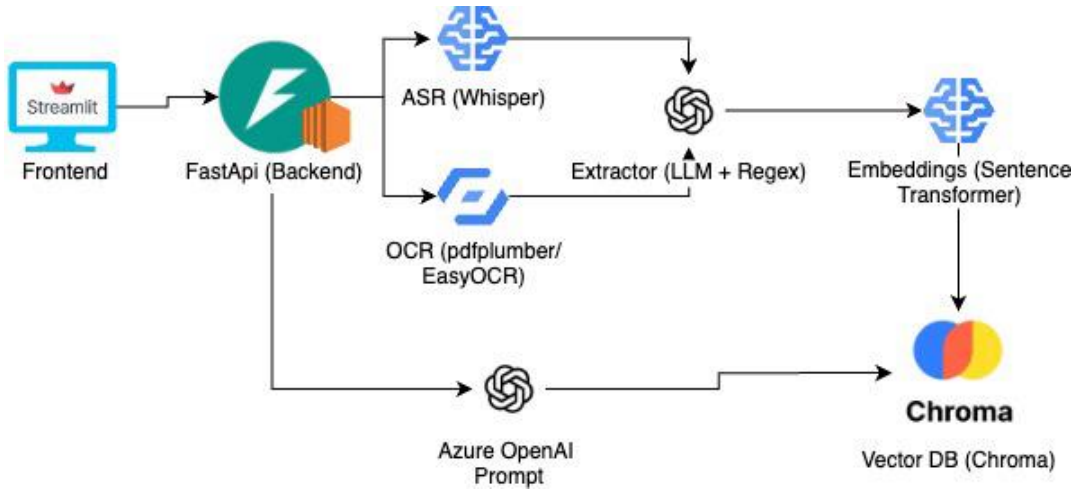
Public

- Arquitectura modular en FastAPI con separación por responsabilidades.
- Uso de embeddings semánticos propios para mejorar la búsqueda contextual.
- Validación de inputs y control de errores.
- Documentación automática de endpoints vía Swagger y ReDoc.
- Compatibilidad multi-arch en el Dockerfile para correr en Intel y ARM (Apple Silicon).

B. Arquitectura propuesta

Diagrama general del sistema

A continuación se muestra el diagrama de alto nivel de la solución:



Componentes del backend y flujo de datos

1) **Ingesta:** /upload_audio y /upload_doc reciben archivos, los guardan en disco (volumen Docker) y desencadenan el pipeline.
2) **Procesamiento de Audio:** app/asr.py aplica faster-whisper y devuelve la transcripción.
3) **Procesamiento de PDF/Imagen:** app/ocr.py intenta sacar texto con pdfplumber y si no hay, usa EasyOCR (imágenes directas o tras convertir páginas de PDF a imagen).
4) **Extracción de entidades:** app/extract.py primero intenta con LLM siguiendo un esquema JSON forzado; si falla, aplica regex para name/age/date y lexicón de síntomas. Se valida con Pydantic.
5) **Indexación:** app/rag.py divide el texto en chunks con overlap, enriquece con metadata (patient_name, date, age) y usa app/storage.py para upsert al vector store.

Public

6) **Consulta:** /chat usa retrieve_chunks (con filtros opcionales), RAG y construye un prompt con contexto para el LLM. Se devuelven citas (source_id) y la respuesta.
7) **Seguridad:** app/security.py implementa autenticación Bearer JWT y autorización por roles; FastAPI Depends protege endpoints.

Justificación de tecnologías y herramientas

- **FastAPI:** rendimiento, tipado, OpenAPI integrado, comunidad madura.
- **Whisper (faster-whisper):** precisión alta y ejecución local eficiente en CPU/GPU.
- **pdfplumber + EasyOCR:** cobertura amplia de PDFs con y sin texto; EasyOCR simplifica despliegue en Docker sin Tesseract.
- **Chroma:** rapidez para POCs/MVPs con persistencia sencilla; sustituible por Qdrant/pgvector en prod.
- **Azure OpenAI/OpenAI:** control de políticas y escalado gestionado; abstracción en app/llm.py permite alternar proveedor.
- **Pydantic:** validación de payloads y esquemas de extracción.
- **Docker + docker-compose:** portabilidad, reproducibilidad y aislamiento de dependencias.
- **sentence-transformers:** Opté por generar nuestros propios embeddings en lugar de depender de un servicio externo (Privacidad, Velocidad y Bajo coste operativo)
- **Streamlit (frontend demo):** velocidad de prototipado, mínimo código, sin necesidad de un framework SPA completo. Ideal para PoC/MVP clínico interno.

Flujo detallado (pipeline RAG)

El pipeline comprende la ingesta de archivos, ASR/OCR, extracción estructurada/no estructurada, chunking con overlap, embeddings y upsert al vector store; luego, en consulta, recuperación semántica + generación con LLM.

C. Plan de desarrollo

¿Qué se entregó en este MVP?

- **Endpoints:**
  - /upload_audio: guarda, transcribe, extrae y indexa.
  - /upload_doc: guarda, extrae texto (OCR) y indexa.
  - /chat: RAG con filtros opcionales por patient_name, date_range y age.
  - /login: login del usuario con autenticación.

Public

- Backend robusto con logging (spans), validación y manejo de excepciones.
- Frontend Streamlit para subir y consultar.
- Seguridad con JWT (dummy users con contraseñas hasheadas) y control de acceso por rol.
- Dockerfile y docker-compose con persistencia de Chroma.

Funcionalidades PLUS implementadas

- Subida de PDFs/Imágenes con OCR.
- Cliente Streamlit.
- Seguridad: autenticación JWT, autorización por roles y hash de credenciales.
- Filtros flexibles en retrieval (AND combinable, rango fechas/edad).
- Citas (source_id) en respuestas para trazabilidad.
- Persistencia local del vector store.

Siguientes pasos recomendados

- **Datos y escalabilidad:** migrar a Qdrant Cloud o Postgres+pgvector; añadir índices por metadata; sharding y HA.
- **Seguridad avanzada:** pasar a un IdP (Auth0/Azure AD), RBAC/ABAC, auditoría, rotación de llaves, rate limiting y WAF.
- **Observabilidad:** OpenTelemetry, métricas (Prometheus), logs centralizados (ELK/CloudWatch), alertas SLO.
- **MLOps:** versionado de datos/modelos (DVC/MLflow), pruebas de regresión, canary releases de prompts/modelos, evaluación RAG.
- **Calidad de datos:** normalización clínica (SNOMED/ICD), deduplicación, matching probabilístico de pacientes.
- Diarización de hablantes y timestamps en ASR.
- Redacción clínica segura (plantillas de SOAP/SOAPIE) y generación controlada.
- End-to-end encryption y almacenamiento seguro de PHI (KMS/Vault).

Ruta a producción (infra, seguridad, MLOps, cloud)

- **Infraestructura:** contenedores en Kubernetes (AKS/EKS/GKE) con HPA; Ingress + TLS; almacenamiento persistente gestionado.
- **Datos:** Qdrant/Qdrant Cloud o Postgres+pgvector; backups automáticos; cifrado en reposo y en tránsito.
- **CI/CD:** GitHub Actions/GitLab CI, escaneo de imágenes (Trivy), IaC con Terraform; despliegues blue/green.

Public

- **Seguridad:** secret management (AWS Secrets Manager/Azure Key Vault), policies de acceso mínimas, auditoría completa.
- **MLOps:** tracking de experimentos, monitoreo de deriva, datasets de evaluación, feedback loop supervisado.
- **Costos:** autoescalado, colas de trabajo (Celery/RQ) para OCR/ASR pesados, cache de embeddings.

Estructura de archivos y módulos principales

- **app/main.py:** Entrypoint FastAPI; define /upload_audio, /upload_doc, /chat y warmup.
- **app/asr.py**: Carga y ejecuta faster-whisper (ASR).
- **app/ocr.py:** pdfplumber + EasyOCR para PDF/imagen; fallback si no hay texto embebido.
- **app/extract.py:** Extracción con LLM (Azure/OpenAI) y fallback regex; Pydantic para validar.
- **app/rag.py:** chunking, index_transcript y retrieve_chunks; filtros por patient_name, date_range y age.
- **app/storage.py:** Persistencia en Chroma; upsert/query; abstracción del vector store.
- **app/llm.py:** Cliente genérico de chat para LLM (Azure/OpenAI) y parámetros de inferencia.
- **app/security.py:** JWT, roles y dummy DB de usuarios (hash de contraseñas, tokenización). Se enfatiza que para producción se debe usar una base de datos real (SQL/NoSQL) e IdP.
- **app/models.py:** Pydantic models (UploadResponse, ChatQuery, ChatResponse, etc.).
- **app/logging_utils.py:** spans y logger centralizado.
- **frontend/app.py:** UI para subida y chat.
- **Dockerfile / docker-compose.yml:** despliegue reproducible, volúmenes persistentes y variables de entorno.

Docker y despliegue local

El Dockerfile utiliza python:3.11-slim, instala dependencias del sistema (ffmpeg, poppler) y Python, copia el proyecto y expone FastAPI en 0.0.0.0:8000. Con docker-compose se definen servicios para backend, frontend y un volumen persistente para los datos de Chroma.

Public

Seguridad (detalle)

- **Autenticación:** Bearer JWT con expiración configurable; generación de tokens al autenticar usuarios dummy.
- **Autorización:** Depends(require_roles(...)) en FastAPI para proteger rutas sensibles.
- **Cifrado:** hash de contraseñas con bcrypt; recomendación de TLS con un reverse proxy (nginx/ingress) en prod.
- **Datos sensibles:** evitar incluir PHI en logs; sanitización de prompts; auditoría y control de accesos.

Conclusión

La solución entrega un MVP funcional y extensible para gestión de conocimiento clínico con RAG multimodal. Se priorizó la simplicidad operacional (ASR/OCR locales, Chroma persistente) y una arquitectura que permite migrar a componentes gestionados en producción (Qdrant, Azure Speech, IdP corporativo). El diseño modular y los contratos claros facilitan la evolución hacia mayores niveles de seguridad, observabilidad y escalabilidad.