Tecnológico de Monterrey

Sistema de análisis de llamadas

Documento de Especificación del Diseño

HowlX
A01285221 - Alejandra Coeto Sánchez
Diego de Jesús Esparza Ruiz
Jesús Adrián López Gaona
Luis Gerardo Juárez García
Mónica Soberón Zubía
Santiago De la Riva Juárez

v1.02
13/06/2025

Control de Versiones

Fecha	Versión	Descripción del cambio	Autor(es)	Revisor(es)
02/05/2025	1.0	Se agregó el diagrama principal y vista de procesos	Alejandra Coeto …	
04/05/2025	1.01	Se agregó la justificación	Alejandra Coeto …	
13/06/2025	1.02	Actualización del diagrama de despliegue	Jesús Adrián Lóp…	Alejandra Co…

Contenido

1. Introducción

1.1. Propósito

El propósito de este sistema es convertir conversaciones de voz en información útil para supervisores, equipos de calidad y analistas de experiencia del cliente. Lo hace mediante una arquitectura fullstack que combina transcripción automática, análisis NLP y generación de reportes visuales. La solución está compuesta por un frontend moderno en Next.js y un backend en Python especializado en IA.

1.2. Objetivos

- Proveer una aplicación web responsiva e intuitiva.
- Automatizar la transcripción de llamadas desde archivos de audio.
- Detectar temas clave, emociones y sentimientos mediante NLP.
- Generar dashboards y reportes exportables.
- Asistir con recomendaciones personalizadas usando modelos LLM (Gemini, OpenAI).
- Permitir autenticación segura de usuarios y control de sesiones.
- Ejecutar pruebas automáticas end-to-end para asegurar calidad.

1.3. Alcance y restricciones

- Frontend en Next.js con autenticación y persistencia de sesiones.
- Backend Python con módulos especializados para IA local y servicios externos.
- Almacenamiento de usuarios y metadatos con PostgreSQL + Prisma.
- Exportación en múltiples formatos y carga de archivos o grabación desde el dispositivo.
- Pruebas automatizadas con Cypress para garantizar estabilidad.

Restricciones:

- Los procesos de IA son exclusivamente backend en Python.
- Dependencia de servicios externos para LLM (Gemini, OpenAI).
- El sistema requiere conectividad para algunas funciones críticas (ej. modelos externos).

1.4. Contexto

En entornos de atención al cliente, se generan grandes volúmenes de llamadas que usualmente no se analizan por falta de herramientas técnicas. Este sistema busca cubrir ese problema con una plataforma web que analiza cada llamada a profundidad, detectando emociones, riesgos y temas comunes y devolviendo un feedback con el que se pueda aportar valor tanto a los operadores como a los administradores. El procesamiento es delegado a un backend robusto en Python, y la interfaz a un frontend moderno.

1.5. Estrategia de solución

Tecnologías

Frontend (Next.js)

- Framework: Next.js
- Lenguaje: TypeScript
- UI: React + TailwindCSS
- Autenticación: NextAuth.js + Prisma + PostgreSQL
- Pruebas E2E: Cypress

Backend (Python)

- Framework: FastAPI
- Transcripción: Whisper (local)
- Análisis NLP: Gemini
- Asistente IA: Integración con APIs de OpenAI, Gemini vía OpenRouter
- Persistencia: PostgreSQL (compartido con frontend)
- APIs: RESTful

Descomposición y responsabilidades

Componente	Responsabilidad Principal
Frontend (Next.js)	UI, carga de archivos, visualización, chatbot
Backend (Python)	Procesamiento de audios, análisis NLP, IA
Prisma ORM	Modelado y acceso a base de datos desde Next.js
PostgreSQL	Almacenamiento de usuarios, llamadas y reportes
Cypress	Validación end-to-end del sistema

Tabla 1: Componentes y responsabilidades.

Atributos de calidad

- **Escalabilidad**: backend desacoplado del frontend, procesamiento por lotes.
- **Seguridad**: autenticación con sesiones cifradas, validación estricta.
- **Mantenibilidad**: backend modular, frontend con componentes reutilizables.
- **Pruebas**: Cypress garantiza regresiones mínimas y estabilidad en despliegues.
- **Experiencia de usuario:** diseño responsive y fluido.

Metodología

Enfoque ágil con sprints cortos, integración continua, pruebas automatizadas y demos frecuentes. Se emplean herramientas de control de versiones (Git), pipelines CI/CD y documentación automática.

1.6. Resumen

El sistema se compone de:

- Frontend en Next.js/React/TS, que gestiona la interacción, autenticación, visualización y envío de archivos.
- Backend en Python, que maneja todo el procesamiento de IA (transcripción, NLP, integración con modelos LLM).
- Prisma como ORM entre Next.js y PostgreSQL.
- Cypress para pruebas automáticas end-to-end.
- Una arquitectura desacoplada que permite escalar el backend sin afectar la interfaz.

2. Contenido detallado

2.1. Identificación de involucrados

Stakeholder	**Método de comunicación**	**Frecuencia**	**Responsable**	**Objetivo**	**Notas**
Elvia Rosas	Teams, presencial	Semanal	Todo el equipo	Recibir retroalimentación y realizar dudas	
Juan Carlos Lavariega	Teams, presencial	4 veces a la semana	Todo el equipo	Recibir retroalimentación y realizar dudas	
Thalia Juarez, Equipo Neoris	Teams, presencial	1 vez cada 2 semanas	Todo el equipo	Recibir retroalimentación y realizar dudas	

Tabla 2: Descripción de stakeholders.

2.2. Diagramas y vistas

2.2.1. Diagrama de despliegue

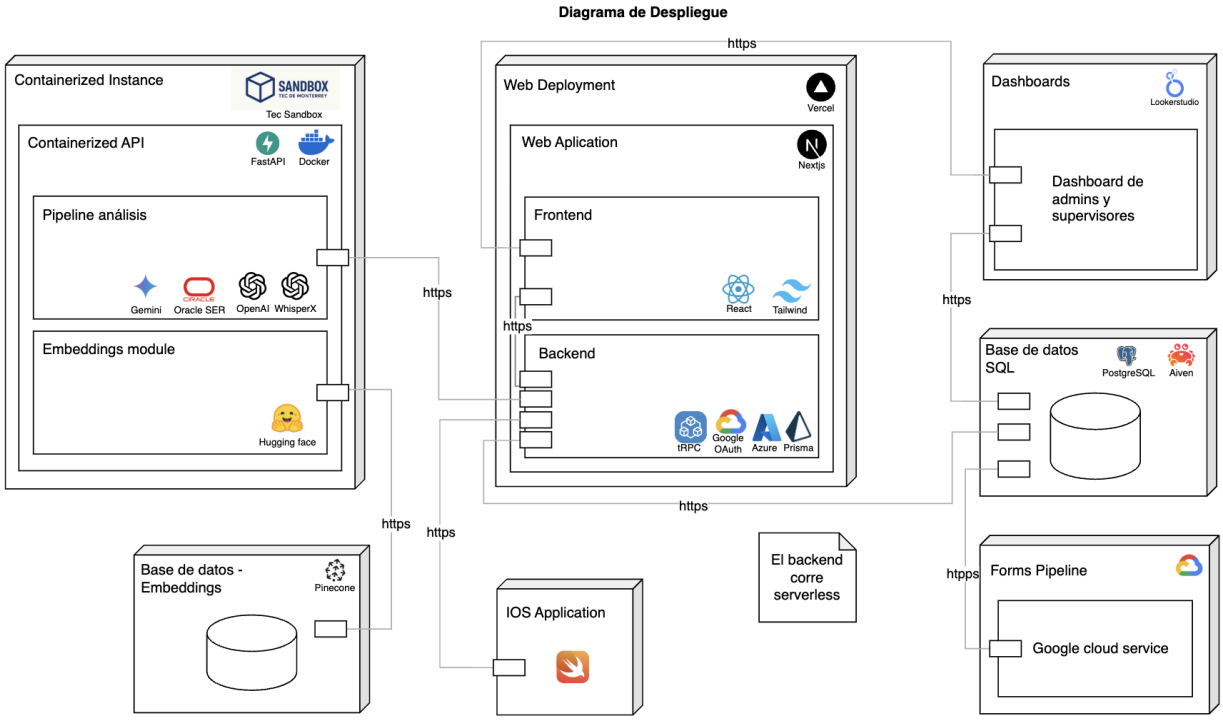

Imagen 1: Diagrama de despliegue con stack usado.

2.2.2. Diagrama de descomposición de subcomponentes - jesus

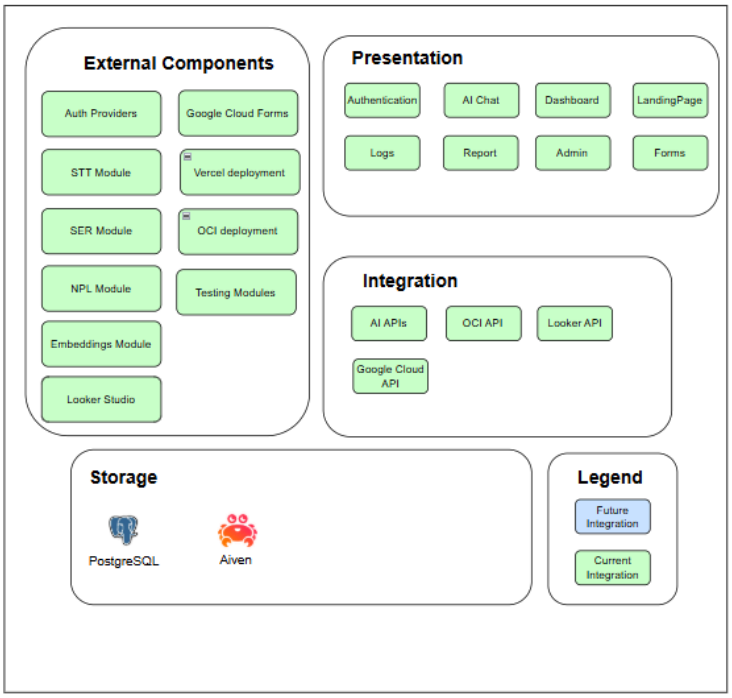

Imagen 2: Diagrama de subcomponentes

2.2.3. Vista Lógica

Este servicio está contenedorizado y desplegado en OCI (Oracle Cloud Infrastructure). Expone el endpoint /upload, que recibe archivos de audio enviados desde el frontend. Una vez recibido el archivo, se inicia una pipeline de procesamiento de inteligencia artificial, la cual puede seguir una configuración por defecto o parámetros personalizados definidos por el usuario.

Pipeline de IA (contenida dentro del backend)

La pipeline es modular y flexible, permitiendo integrar distintas herramientas según las necesidades del cliente:

- Transcripción: WhisperX y huggingface, o Gemini
- Análisis de emociones sobre el texto: OCI sentiment análisis o Gemini
- Análisis de emociones sobre voz: LLM (puede ser personal o publica)
- Generación de reporte: LLM (puede ser personal o publica)

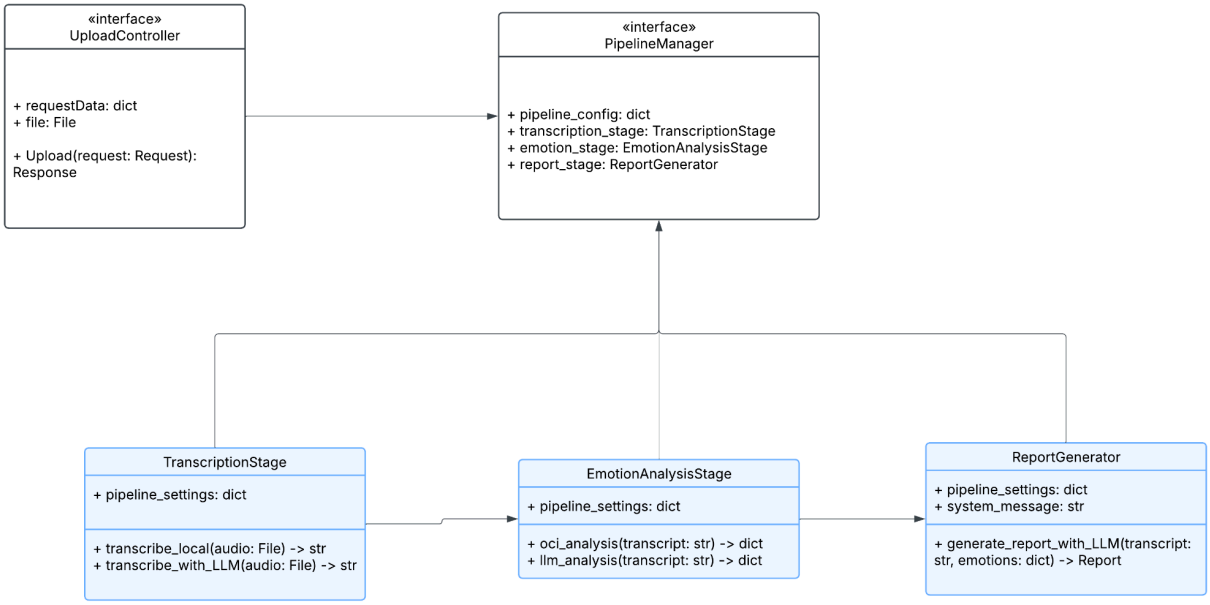

Imagen 3: Diagrama de vista lógica de AI pipeline

2.2.4. Vista de Processos

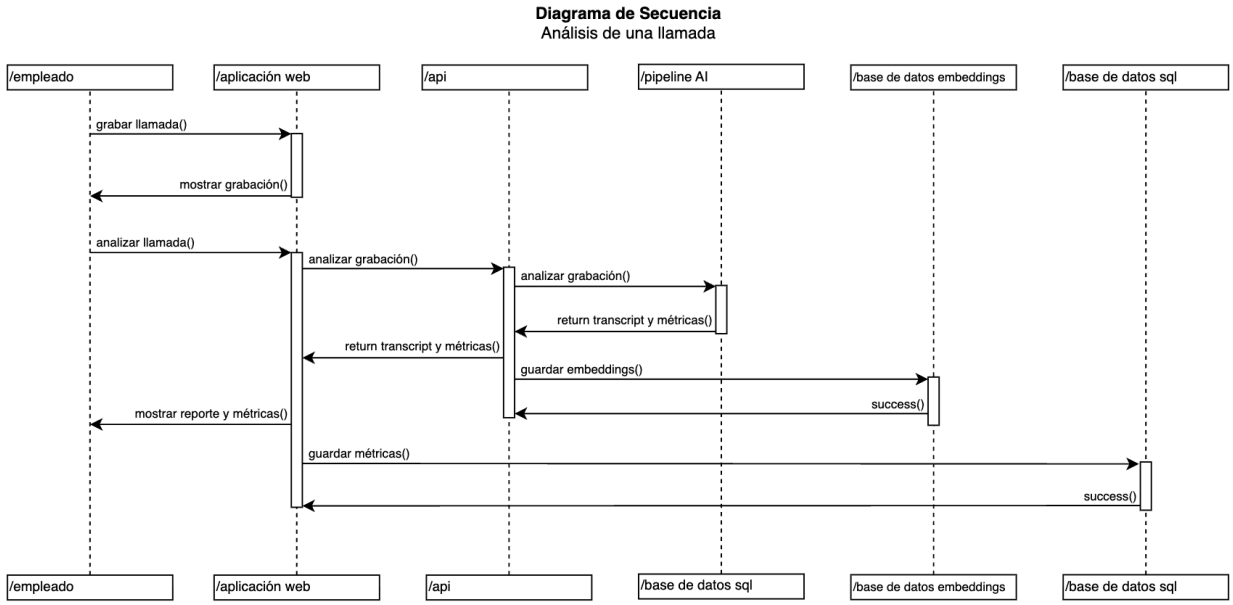

Imagen 4: Diagrama de secuencias de la grabación y análisis de una llamada.

3. Justificación

Para la implementación del sistema, se decidió hacer una aplicación web (la cual contiene tanto front-end como back-end) y una API de python para el procesamiento de inteligencia artificial. Adicionalmente, se utilizan dos bases de datos. La primera (sql) es accedida únicamente mediante el backend de la aplicación web, mientras que la segunda base de datos vectorial es accedida únicamente mediante la API. Esto se decidió para limitar el acceso a cada una de las bases de datos y tener mayor control de quienes modifican o acceden a los datos. Adicionalmente, desde la página web se utilizó un ORM llamado prisma para acceder a la base de datos y también tenía más sentido que el acceso a la base de datos sea desde la app ya que la mayoría de los cambios o modificaciones se realizarían a partir de la aplicación, y hacerlo desde la API sería más lento al tener que enviar la información por más medios. De la misma manera, la base de datos vectorial únicamente es usada por el módulo de embeddings, por lo cual la conexión se realizó por la API que cuenta con los módulos de AI y el de embeddings.

Por otro lado, tanto el front-end y back-end corren desde la misma aplicación y se comunican mediante una API de tRPC (para realizar procedimientos remote procedure call), la cual permite mantener una mayor integridad de datos al mantener esquemas de datos que coinciden tanto en el front como en el back y agrupan los procedimientos acorde a la clase o tabla (en sql).

Adicionalmente, se integraron otros servicios como el de Google Cloud para ejecutar comandos al contestar un forms y el dashboard de Looker Studio, los cuales son independientes pero sí se conectan directamente a la base de datos para actualizarse en tiempo real. Además el dashboard únicamente lee

los datos, mientras que el servicio del formulario agrega datos a una tabla que solo es usada por ese servicio y leída por la app web.

Finalmente, se optó por realizar el procesamiento del análisis de las llamadas en python debido a que cuenta con varias librerías que permiten utilizar distintos tipos de modelos de manera rápida y eficiente.

Además, se decidió implementar una pipeline modular para el análisis de audio, con el objetivo de proporcionar al cliente una solución flexible que combine eficiencia y privacidad. Para lograrlo, se diseñó una arquitectura contenedorizada con Docker, basada en un backend desarrollado en Python. Este backend permite integrar herramientas de inteligencia artificial como WhisperX para transcripción, modelos de Hugging Face para análisis de emociones y sentimientos, y LLMs populares como OpenAI o Gemini para generación de insights adicionales. Cada etapa del pipeline (transcripción, análisis emocional, reporte final) puede configurarse individualmente, de modo que el cliente pueda elegir qué tecnología utilizar según sus propias prioridades: privacidad, rendimiento o precisión. Gracias al uso de FastAPI, fue posible estructurar esta solución de forma clara, escalable y compatible con múltiples módulos, permitiendo al cliente personalizar la experiencia de análisis de llamadas según sus necesidades. FastAPI se hospeda dentro de un contenedor desplegado en OCI (Oracle Cloud Infrastructure), el cual expone un endpoint /upload que permite la interacción directa con el frontend. A través de este endpoint, los usuarios pueden cargar archivos de audio para ser transcritos y analizados, ya sea utilizando una configuración predeterminada o parámetros personalizados según sus necesidades. Esto habilita un flujo continuo entre la interfaz web y el motor de análisis en backend, asegurando una experiencia fluida y adaptable para el usuario final.