Prueba Técnica — Full-Stack Node.js + Next.js (AI-first)

Propósito

Queremos evaluar tu capacidad para **diseñar y desarrollar** un producto funcional end-to-end con un enfoque en:

- **Arquitectura**: modular, escalable, mantenible.
- **DevOps**: despliegue, trazabilidad, observabilidad, CI/CD.
- **Uso de IA**: integración práctica y con valor real en el flujo.
- **Buenas prácticas**: manejo de errores, logs estructurados, seguridad por defecto.

La prueba no busca que sigas instrucciones exactas, sino que **demuestres cómo planteas y ejecutas** una solución.

Tiempo estimado: 4**–8 horas** (dentro de un plazo máximo de 72 h).

Contexto del reto

Diseña y construye un **micro-producto** que permita:

1. Un backend en **Node.js/TypeScript** con APIs claras, validación, logs, métricas y persistencia.
2. Un frontend en **Next.js (App Router)** con SSR o acciones de servidor, integrando el backend.
3. Integración de **IA** que aporte valor funcional real (no solo de demostración).
4. Trazabilidad y observabilidad (IDs de correlación, métricas básicas).
5. Despliegue funcional (idealmente en **Vercel** para el front y en un servicio gestionado o contenedor para el back).

Caso de uso propuesto

Signal Watcher

Un analista crea “listas de observación” con términos (por ejemplo, nombres de marca, dominios o palabras clave).

El sistema simula recibir eventos (p. ej., “nuevo dominio sospechoso detectado”) y la IA debe:

- **Resumir** el evento en lenguaje natural.
- **Clasificar** su severidad (LOW / MED / HIGH / CRITICAL).
- **Sugerir** la siguiente acción para el analista.

Puedes implementar la IA con un proveedor real (OpenAI, Azure OpenAI, etc.) o diseñar un adaptador con **modo mock** para pruebas.

Aspectos clave a demostrar

Backend

- API en Node.js + TypeScript (Express o Fastify).
- Modelado y persistencia (PostgreSQL + Prisma recomendado).
- Cache para resultados recientes (Redis recomendado).
- Logs estructurados con IDs de correlación y métricas simples.
- Middleware de manejo de errores con códigos HTTP adecuados.
- Configuración segura vía variables de entorno.

Frontend

- Next.js (App Router) con SSR o Server Actions.
- Interfaz para crear listas, simular eventos y enriquecimiento por IA.
- Manejo de estados de carga y errores.
- Estilos y componentes coherentes (puedes usar Tailwind/shadcn).

Arquitectura y DevOps

- Estructura de carpetas modular y clara (separación por capas y responsabilidades).
- Pipeline CI/CD básico (build, test, lint).
- Despliegue funcional y accesible (Vercel para el front; cualquier opción razonable para el back).
- Documentación mínima viable:
  - **README**: cómo correr y desplegar.
  - **ADR**: decisiones técnicas clave.
  - **Runbook**: tareas de operación y soporte.

Entregables

1. **Repositorio** con todo el código y documentación.
2. `.env.example` sin secretos reales.
3. **PROMPT_LOG.md** con el historial resumido de uso de IA.
4. Capturas o gif corto del flujo principal.
5. URL del frontend desplegado (y del backend si aplica).

Evaluación

- **Arquitectura y claridad del código** (estructura, patrones, mantenibilidad).
- **Calidad de DevOps** (despliegue, CI/CD, trazabilidad, observabilidad).
- **Integración y valor real de la IA**.
- **Buenas prácticas** (validación, seguridad, manejo de errores).
- **Documentación** y facilidad para ejecutar el proyecto.