DESAFÍO TÉCNICO – SEMI SENIOR

Concurrencia, Manejo de Recursos y Worker Threads

Objetivo

Evaluar la capacidad del candidato para trabajar con concurrencia, optimizar el uso de recursos del sistema, coordinar tareas costosas usando **Worker Threads** y mantener una interfaz en tiempo real mediante **Socket.io** y **React** (o framework a elección).

Descripción General

El candidato deberá desarrollar una aplicación conformada por: Backend en **Node.js** utilizando Worker Threads. Frontend en **React** o framework moderno a elección. Comunicación en tiempo real mediante **Socket.io**. El sistema ejecutará y monitoreará tareas costosas asignadas a workers, asegurando una correcta gestión de memoria, reemplazo dinámico de workers finalizados y estabilidad con un máximo de **5 workers activos** en todo momento.

1. Inicialización de Tareas

El sistema debe iniciar con **5 tareas activas**. Cada tarea tiene una duración X configurable (ejemplo: entre 10 y 40 segundos). Cada tarea debe ejecutarse dentro de un Worker Thread independiente.

2. Monitoreo en Tiempo Real

El backend debe enviar información en tiempo real al frontend, incluyendo: Uso de **CPU** Uso de **Memoria RAM** Estado de cada worker (**activo** / **inactivo**) Tiempo transcurrido ID del worker El frontend debe representarlo mediante una **tabla o panel de monitoreo** actualizada en tiempo real.

3. Finalización y Reemplazo Automático de Tareas

Cuando un worker termina su tarea, su estado debe cambiar a **inactivo**. El backend debe enviar el **tiempo total de ejecución**. Inmediatamente debe destruir el worker para liberar memoria. Un nuevo worker debe generarse automáticamente para mantener siempre exactamente **5 activos**. El frontend debe reemplazar la información del worker terminado sin duplicar registros.

4. Manejo Correcto de Recursos

Este es uno de los criterios más importantes. Se evaluará: Destrucción correcta de workers al finalizar. Ausencia de fugas de memoria. Evitar workers huérfanos o procesos sin control. Uso eficiente de CPU y memoria. Estructura clara para administración del pool de workers.

5. Consideraciones Técnicas

El backend debe soportar reemplazos continuos sin degradación. El frontend debe mantenerse estable y fluido. Debe evitarse cualquier duplicación de workers en pantalla. El código debe estar modularizado usando buenas prácticas.

6. Entregables

Repositorio con frontend y backend. README con instrucciones claras. Explicación del enfoque adoptado para: Concurrencia Monitoreo Manejo de recursos Diseño del Worker Manager

Criterios de Evaluación

Concurrencia y paralelismo bien implementados. Manejo eficiente de Worker Threads. Monitoreo estable y exacto. Correcta liberación de recursos. Estabilidad del sistema a largo plazo. Legibilidad, limpieza y mantenibilidad del código. Interfaz clara y funcional.