□ Desafío 5: Orquestador de Tareas Distribuidas con Node.js

□ Enunciado

Diseñá y desarrollá un sistema en **Node.js** capaz de recibir, distribuir y monitorear tareas pesadas en paralelo, utilizando el módulo `worker_threads`.
El sistema debe actuar como un **orquestador de tareas distribuido**, donde un proceso principal (Master) asigna trabajos a un conjunto de workers (hilos) que ejecutan las tareas y reportan su progreso.

□ Requisitos Principales

1. **Gestión de tareas:**
   Cada tarea tiene un **ID**, **tipo**, **prioridad** y **payload**.
   Las tareas se encolan y se procesan según su prioridad.
   Las tareas fallidas deben **reintentarse automáticamente** hasta un número máximo configurable.

2. **Manejo de workers:**
   El sistema debe mantener un **pool dinámico de workers**.
   Si la cola crece, se agregan más workers; si disminuye, se eliminan.
   Debe existir un **mecanismo de heartbeat** o monitoreo para detectar workers colgados o inactivos.

3. **Comunicación:**
   Los workers se comunican con el proceso principal usando `MessageChannel` o `postMessage`.
   El Master envía tareas a los workers y recibe resultados, errores o notificaciones de progreso.

4. **Persistencia y métricas:**
   El sistema debe guardar el estado de las tareas (**pendientes**, **en proceso**, **completadas**, **fallidas**).
   Debe exponer un endpoint o interfaz que muestre **estadísticas en tiempo real** (cantidad de workers activos, tareas pendientes, completadas, tiempo de ejecución, etc.).

5. **Escalabilidad y resiliencia:**
   Si un worker muere, sus tareas deben **reasignarse automáticamente**.
   El diseño debe permitir extender el sistema para que funcione con múltiples instancias del proceso principal en el futuro (**escalado horizontal**).

□ Dashboard de Monitoreo — *Anexo Técnico*

□ Objetivo

Visualizar en tiempo real el estado del orquestador, los workers activos, las tareas y las métricas del sistema.

□ Secciones Principales

A. Estado General del Sistema

- Estado del sistema: □ *Stable*, □ *High Load*, □ *Recovering*
- Tareas totales
- Workers activos
- Cola de tareas pendientes
- Promedio de ejecución
- Última actualización (`hh:mm:ss`)

B. Tabla de Workers Activos

Worker ID	Estado	CPU (%)	RAM (MB)	Tareas Asignadas	Último Heartbeat
worker-1	□ Activo	12%	150	3	hace 2 seg
worker-2	□ Cargando	35%	230	5	hace 1 seg

□ **Endpoint sugerido:** `/workers/status`
Actualiza la información cada **2–5 segundos** mediante polling o WebSocket.

C. Cola de Tareas

ID Tarea	Tipo	Prioridad	Estado	Retries	Tiempo Ejecución (ms)	Asignado a
task-001	Procesamiento	Alta	⏳ En proceso	0	1,204	worker-1
task-002	Transformación	Media	✅ Completada	0	958	worker-2

Endpoints sugeridos:

- `GET /tasks` → lista todas las tareas
- `GET /tasks?status=pending` → filtra por estado
- `GET /tasks/:id` → detalle de una tarea específica

D. Métricas en Tiempo Real (Gráficos)

- Uso de CPU y memoria por worker (líneas o barras)
- Cantidad de tareas procesadas por minuto (área)
- Distribución de tipos de tarea (circular)

💡 **Sugerencia:** Usar `Socket.IO` o `EventSource (SSE)` para recibir métricas sin refrescar la página.

E. Historial de Eventos

Timestamp	Evento	Descripción
10:21:02	Worker creado	`worker-5` añadido al pool
10:21:05	Tarea iniciada	`task-112` asignada a `worker-3`
10:21:10	Tarea fallida	`task-104` reintentando (2/3)

⚙ Stack Sugerida

- **Backend:** Node.js (Express + worker_threads + Redis o memoria local)
- **Frontend:** React + Chart.js o Recharts
- **Comunicación:** WebSocket o SSE
- **Persistencia:** SQLite o Redis

📡 Ejemplo de Endpoint de Métricas Globales

```
{
  "systemStatus": "Stable",
  "workers": {
    "active": 5,
    "idle": 1
  },
  "tasks": {
    "pending": 12,
    "inProgress": 8,
    "completed": 73,
    "failed": 3
  },
  "averageExecutionTimeMs": 1175,
  "uptime": "02:34:55"
}
```