## **Evaluación Técnica - Node.js**

### **Objetivo**

Implementar una API REST que procese tareas de forma concurrente usando **Worker Threads** y aplique correctamente patrones de diseño.

**Tiempo Estimado: 1 Hora**

## **Requisitos Técnicos**

* **NodeJS 18+**
* **JSDoc**
* **Procesos en segundo plano**
* **Patrones de Diseño**

## **Descripción del Ejercicio**

**Se debe construir una aplicación con:**

1. Implementar el worker.js para manejar tareas intensivas.
2. Identificar y corregir el bug en /buggy-endpoint (problema con el Event Loop).
3. Uso de uno o más patrones: **Strategy, Singleton e Inyección de Dependencias**.

## **Desarrollo**

/\* eslint-disable no-console \*/

/\*

### Prueba Técnica Node.js - Senior Backend Developer

Objetivo: Implementar una API que procese cálculos matemáticos pesados usando \*\*Worker Threads\*\*. Debe usar:

- Patrón \*\*Strategy\*\* para definir diferentes estrategias de cálculo.

- \*\*Inyección de Dependencias\*\* para manejar las estrategias.

- Patrón \*\*Singleton\*\* para la gestión de workers.

- \*\*Corrección de un bug en el Event Loop\*\*.

\*/

/\*

---

### \*\*Tareas del Candidato:\*\*

1. Implementar el \*\*worker.js\*\* para manejar cálculos intensivos.

2. Identificar y corregir el bug en `/buggy-endpoint`.

3. Explicar cómo el uso de \*\*Strategy, Singleton e Inyección de Dependencias\*\* se aplica en esta solución.

\*/

// eslint-disable-next-line max-classes-per-file

import { Worker } from 'worker\_threads';

import express from 'express';

// Singleton para manejar Workers

class WorkerPool {

// Construir este metodo

executeTask(workerFile, data) {

}

}

const workerPool = new WorkerPool();

// Strategy Pattern - Diferentes cálculos matemáticos

class SumStrategy {

execute(a, b) {

return a + b;

}

}

class MultiplyStrategy {

execute(a, b) {

return a \* b;

}

}

class Calculator {

strategy;

constructor(strategy) {

this.strategy = strategy;

}

calculate(a, b) {

return this.strategy.execute(a, b);

}

}

// Express API

const app = express();

app.use(express.json());

app.post('/calculate', async (req, res) => {

const { a, b, operation } = req.body;

let strategy;

if (operation === 'sum') {

strategy = new SumStrategy();

} else if (operation === 'multiply') {

strategy = new MultiplyStrategy();

} else {

return res.status(400).json({ error: 'Invalid operation' });

}

const calculator = new Calculator(strategy);

const result = calculator.calculate(a, b);

return res.json({ result });

});

// Endpoint para cálculo intensivo usando Worker Threads

app.post('/heavy-task', async (req, res) => {

try {

const result = await workerPool.executeTask('./worker.js', req.body.number);

res.json({ result });

} catch (error) {

res.status(500).json({ error: error.message });

}

});

// \*\*BUG EN EL EVENT LOOP\*\* (Debe ser corregido por el candidato)

app.get('/buggy-endpoint', async (req, res) => {

console.log('Start request');

let stop = false;

setTimeout(() => {

console.log('Timeout executed');

stop = true;

}, Math.random() \* 100);

while (!stop) {

const now = Date.now();

while (Date.now() - now < 5) { }

}

res.send('Response reached... or maybe not 🤔');

});

app.listen(3000, () => console.log('Server running on port 3000'));

## **Criterios de Evaluación**

| **Requisito** | **Evaluación** |
| --- | --- |
| Estructura del Código y Buenas Prácticas | Uso adecuado de módulos, separación de responsabilidades y manejo de errores. |
| Inyección de Dependencias | Usa un **contenedor de dependencias** o un mecanismo de **inyección de dependencias** correctamente implementado. |
| Uso de Workers y Procesos en Segundo Plano | Separa bien las tareas de CPU intensivas del Event Loop principal. |
| Resolución del Bug | Identifica correctamente el problema en el código buggy y justifica de su solución con una explicación técnica clara. |

## **¿Cómo Evaluarlo?**

* ✅ **Comprensión de los patrones de diseño y principios de arquitectura en Node.js.**
* ✅ **Habilidad para estructurar código de manera modular y escalable.**
* ✅ **Manejo de procesos en segundo plano y optimización del Event Loop.**
* ✅ **Capacidad para encontrar y corregir errores complejos.**
* ✅ **Buenas prácticas en pruebas y manejo de errores**

Si tienes dudas o preguntas con respecto a la evaluación, puedes comunicarte con el equipo de tecnologías de oncredit.