TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR

DESARROLLO WEB

MANEJO DE PROCESOS ASÍNCRONOS EN NODE.JS

MANEJO DE PROCESOS ASÍNCRONOS EN NODE.JS: PROMESAS Y CALLBACKS

El manejo de procesos asíncronos representa una de las características más distintivas y poderosas de Node.js, permitiendo ejecutar operaciones no bloqueantes que optimizan el rendimiento y la escalabilidad de las aplicaciones. JavaScript tradicionalmente manejaba la asincronía mediante callbacks, funciones que se ejecutan una vez completada una operación asíncrona. Sin embargo, esta aproximación puede llevar al "callback hell" cuando se anidan múltiples operaciones asíncronas. Las Promesas y, posteriormente, async/await han evolucionado como soluciones más elegantes y mantenibles para gestionar la asincronía en Node.js.

Las Promesas representan un objeto que encapsula el resultado eventual de una operación asíncrona, proporcionando una interfaz más limpia y predecible para manejar tanto el éxito como el fallo de estas operaciones. Los métodos .then(), .catch() y .finally() permiten encadenar operaciones asíncronas de manera más legible que los callbacks anidados. Granados La Paz (2023) enfatiza que el manejo eficiente de operaciones asíncronas es crucial en aplicaciones web modernas, donde las interacciones con bases de datos, servicios externos y operaciones de archivo deben ejecutarse sin bloquear el hilo principal de ejecución.

En aplicaciones de ingeniería de software, el manejo adecuado de la asincronía es fundamental para crear sistemas responsivos y escalables. Por ejemplo, en una aplicación de comercio electrónico, las operaciones como verificación de inventario, procesamiento de pagos y envío de notificaciones deben ejecutarse de manera asíncrona para no bloquear la interfaz de usuario. Node.js permite orquestar estas operaciones complejas utilizando Promesas, async/await, o librerías especializadas como Bluebird para casos más avanzados.

Ejercicio práctico. Implementar una función que simule operaciones asíncronas con diferentes enfoques.

Paso 1. Crear archivo con callbacks: function operacionCallback(callback) { setTimeout(() => callback(null, 'Resultado callback'), 1000); }; operacionCallback((err, resultado) => console.log(resultado));.

Paso 2. Implementar con Promesas: function operacionPromesa() { return new Promise((resolve) => setTimeout(() => resolve('Resultado promesa'), 1000)); }; operacionPromesa().then(resultado => console.log(resultado));.

Paso 3. Usar async/await: async function operacionAsync() { const resultado = await operacionPromesa(); console.log(resultado); return resultado; }; operacionAsync();.

Paso 4. Ejecutar y comparar enfoques.

Resultado. Se observa la evolución desde callbacks hasta async/await, mostrando cómo cada enfoque mejora la legibilidad y mantenibilidad del código asíncrono.