Trabajo Práctico: Instalación de herramientas

Actividad

Responder las siguientes preguntas sobre node

1. **¿Qué es Node.js y en qué se diferencia de JavaScript en el navegador?**

   La diferencia entre Node.js y JavaScript es que JavaScript es un lenguaje de programación que se puede aplicar al desarrollo tanto del lado del cliente como también del lado del servidor. En cambio, Node.js es un entorno de ejecución que permite ejecutar JavaScript en el servidor. Como conclusión, JavaScript es un lenguaje, y Node.js es una plataforma que permite la ejecución en el lado del servidor.

2. **¿Qué son las versiones LTS?**

   LTS significa Long/Term Support (Soporte a Largo Plazo). Es un termino usado comunmente en software y tecnología para referirse a las versiones de un S.O., aplicación, o framework que reciben actualizaciones de seguridad y mantenimiento durante un periodo extendido, generalmente de varios anos.

3. **Explica la diferencia entre un proceso síncrono y uno asíncrono**

   Diferencias:
   Porceso Sincrono:
   _ Se ejecuta paso a paso en un orden secuencial
   _ Cada tarea bloquea la ejecución hasta que se completa.
   _ No pasa a la siguiente tarea hasta que la anterior haya terminado

   Proceso Asincrono:
   _Permite ejecutar otras tareas mientras una operación esta en espera
   _No bloquea la ejecución del programa
   _Generalmente usa callbacks, promesas o async/await

   En resumen, un proceso síncrono es como una fila en un banco donde atienden a una persona a la vez y nadie avanza hasta que termine. En cambio, un proceso asincrónico es como pedir comida en un restaurante con beeper (es un aparato electrónico con bocina que cuando tu pedido u orden esta lista te avisa emitiendo un sonido) donde ordenas tu comida, seguís haciendo otras cosas y te avisan cuando esta listo

4. **¿Cómo se maneja la asincronía en Node.js? Explica las diferencias entre callbacks, Promises y async/await.**

   En Nodejs la asincronia se maneja utilizando callbacks, pormises y también async/await.

   _ Callbacks:es Los callbacks fueron el primer método para manejar la asincronía en Node.js. Son funciones que se pasan como argumento a otra función y se ejecutan una vez que la operación asíncrona finaliza

_ Promises: Las Promises fueron introducidas para mejorar la legibilidad del código y manejar mejor los errores. Una *Promise* representa un valor que estará disponible en el futuro, y puede tener tres estados:
. pending (pendiente),
. fulfilled (resuelta con éxito),
. rejected (rechazada con error).

_ Asyn/Await: async/await es una mejora sobre las Promises que permite escribir código asíncrono con una sintaxis más parecida al código síncrono.

En resumen, aunque los callbacks fueron el método original en Node.js, hoy en día async/await es el enfoque preferido por su claridad y facilidad de uso. Sin embargo, entender Promises sigue siendo clave, ya que async/await está basado en ellas

5. ¿Qué es el objeto `global` en Node.js y en qué se diferencia de `window` en el navegador?

En Node.js, el objeto global es `global`, mientras que en el navegador es `window`. La diferencia principal es que `window` representa el entorno del navegador, incluyendo cosas como el DOM, `localStorage` y `fetch`, mientras que `global` en Node.js está más enfocado en el entorno de ejecución del servidor, con cosas como `process`, `__dirname` y `require`. Otra diferencia clave es que en el navegador, las variables globales se agregan a `window`, mientras que en Node.js cada archivo es un módulo separado y las variables no se añaden automáticamente a `global`. Además, Node.js no tiene acceso a APIs del navegador como `alert` o `document`.

6. ¿Node.js es multithreading?

Node.js no es multithreading por defecto en la ejecución de código JavaScript, ya que utiliza un modelo de un solo hilo (*single-threaded*) basado en el event loop. Sin embargo, puede manejar múltiples operaciones concurrentes de manera eficiente gracias a su arquitectura asíncrona y no bloqueante.

Casos donde Node.js usa múltiples hilos:
1_Módulo worker_threads: Permite crear hilos secundarios cuando es necesario.
2_Hilos en el núcleo de Node.js: Algunas operaciones de bajo nivel (como las de fs y crypto) usan hilos del Thread Pool de libuv para tareas intensivas.
3_Cluster Module: Permite ejecutar múltiples procesos de Node.js para aprovechar todos los núcleos del CPU.

A pesar de su naturaleza de un solo hilo, Node.js puede manejar múltiples conexiones simultáneamente mediante operaciones asíncronas, lo que lo hace altamente eficiente para aplicaciones I/O intensivas.

7. **¿Qué es el Event Loop en Node.js y cuál es su papel en la ejecución de código asíncrono?**

   El Event Loop es el mecanismo principal que permite a Node.js ejecutar código de manera asíncrona y no bloqueante en un solo hilo. Su función es manejar operaciones de entrada/salida sin bloquear el flujo de ejecución.

   Fases del Event Loop:
   1_Timers: Ejecuta funciones programadas con setTimeout y setInterval.
   2_Pending Callbacks: Maneja callbacks de operaciones I/O.
   3_Idle, prepare: Interno de Node.js (poco usado directamente).
   4_Poll: Obtiene nuevas operaciones I/O y ejecuta callbacks disponibles.
   5_Check: Ejecuta setImmediate().
   6_Close Callbacks: Maneja eventos de cierre como socket.on('close', ...).

8. **Mencionar tres ventajas de Node.js**

   Ventajas:
   1_Alto rendimiento y escalabilidad: Gracias a su modelo asíncrono y no bloqueante, Node.js puede manejar muchas conexiones simultáneamente sin consumir muchos recursos.
   2_Un solo lenguaje (JavaScript) en toda la aplicación: Se puede escribir tanto el frontend como el backend en JavaScript, lo que facilita el desarrollo y mantenimiento.
   3_Gran ecosistema de paquetes (NPM): Node.js tiene una comunidad activa con más de un millón de paquetes en npm, lo que permite reutilizar código y acelerar el desarrollo.

9. **menciona tres desventajas de Node.js.**

   Desventajas:
   1_No es ideal para tareas CPU-intensivas: Debido a su arquitectura de un solo hilo, Node.js no es la mejor opción para tareas que requieren mucho procesamiento, como el cálculo de algoritmos complejos o el procesamiento de imágenes.
   2_Callback hell y manejo de asincronía: Aunque async/await ha mejorado la situación, Node.js sigue siendo propenso a problemas de callback hell si no se estructura bien el código.
   3_Inestabilidad en dependencias: Muchas librerías en npm son de código abierto y no siempre están bien mantenidas, lo que puede generar problemas de compatibilidad y seguridad.

10. **¿Podrías nombrar algunas bibliotecas que los desarrolladores utilizan frecuentemente con Node.js?**

    Express.js → Framework minimalista para crear APIs y servidores web.

    Mongoose → ODM para trabajar con MongoDB de manera sencilla.

    Bcrypt → Para el hashing de contraseñas.

    Socket.io → Para comunicación en tiempo real con WebSockets.

    Axios → Cliente HTTP para hacer solicitudes a APIs.

Conclusión

Node.js es una plataforma poderosa para construir aplicaciones escalables y eficientes, especialmente en entornos donde la concurrencia y la velocidad de I/O son críticas. A pesar de sus limitaciones en tareas CPU-intensivas, su comunidad, rendimiento y ecosistema de paquetes lo convierten en una opción popular para el desarrollo backend.