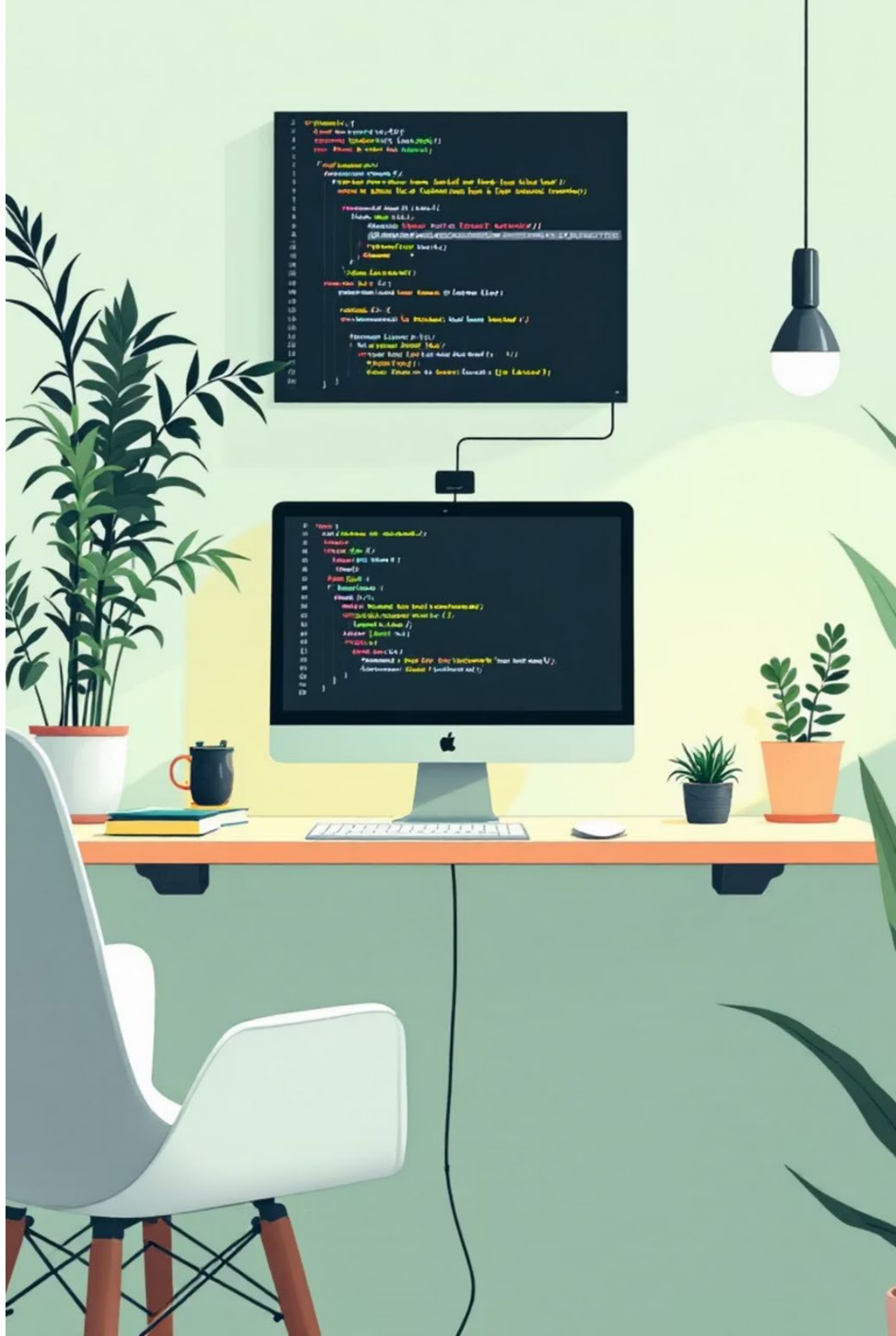
Módulo fs en Node.js

Profesor: González Franco



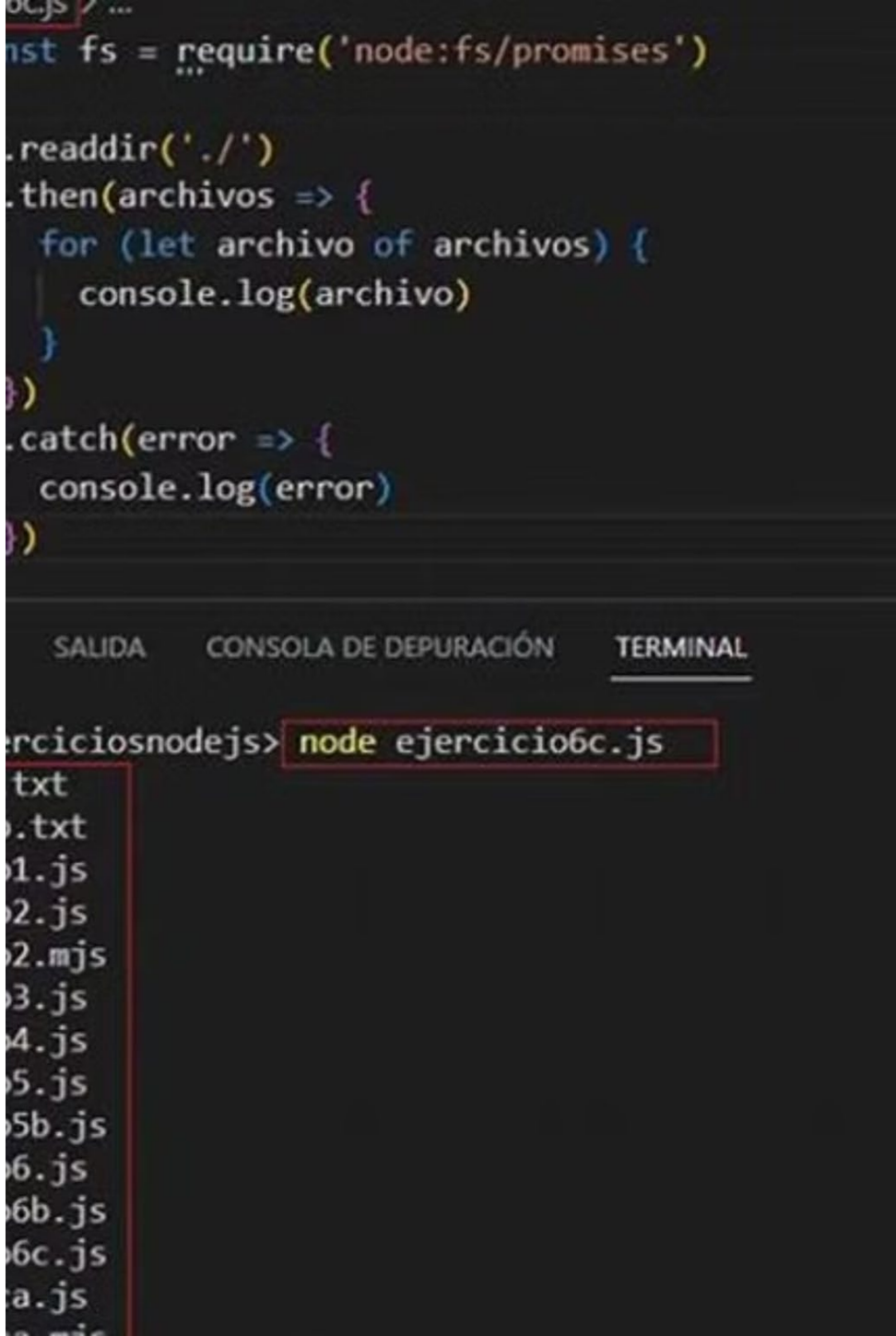
¿Qué es el módulo fs en Node.js?

Sistema de Archivos

El módulo `fs` (File System) es la herramienta principal para interactuar con el sistema de archivos del servidor desde Node.js.

Funcionalidades Clave

Permite realizar operaciones esenciales como leer, escribir, crear y eliminar archivos, facilitando la gestión de datos persistentes.



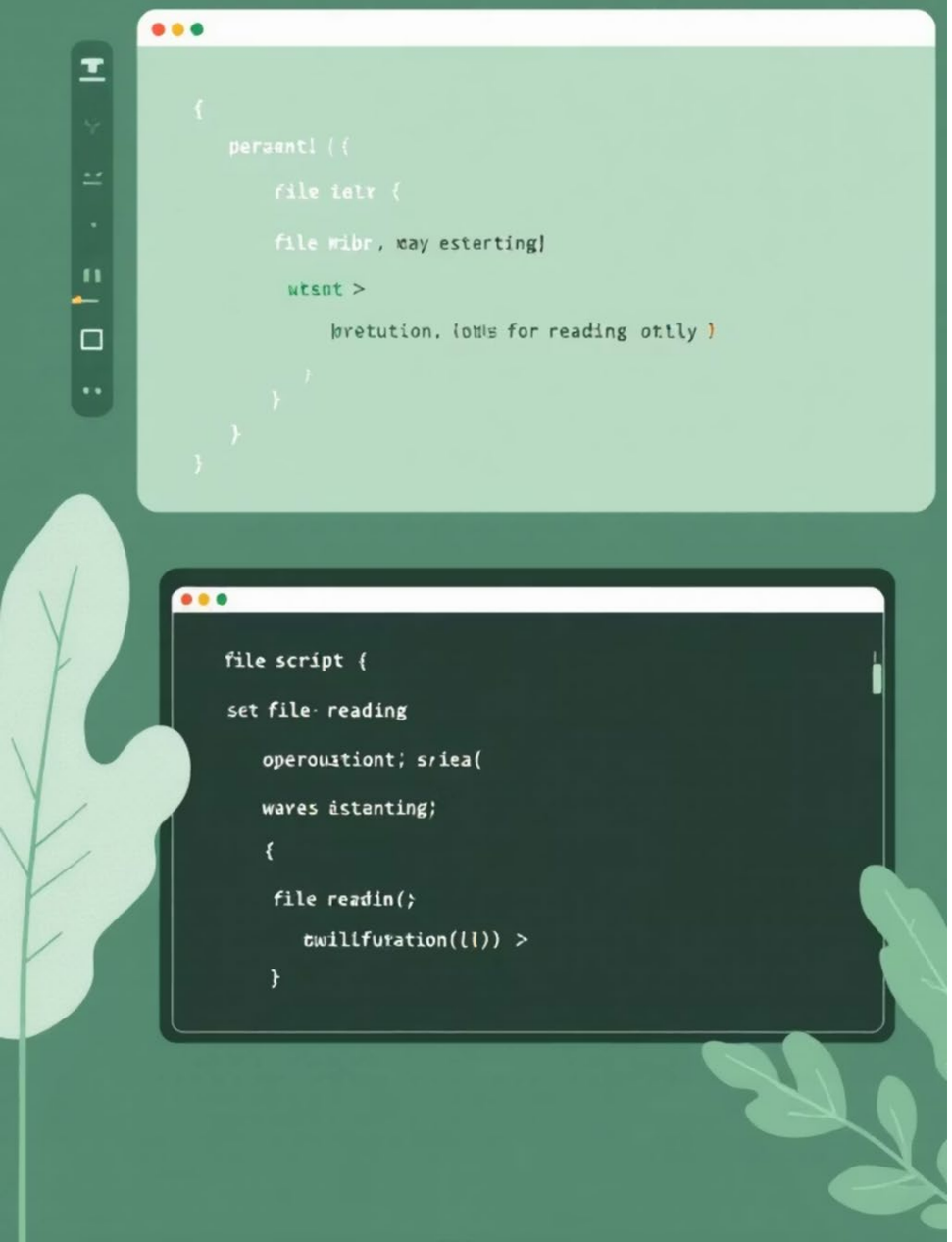
Importancia del módulo fs en el desarrollo

Gestión de Datos Persistentes

El módulo `fs` permite a las aplicaciones mantener datos de forma permanente, facilitando la lectura y escritura de archivos para preservar el estado y configuración.

Integración de Datos Externos

Capacidad para cargar y manipular archivos de diferentes formatos como JSON, CSV y TXT, mejorando la flexibilidad en el intercambio de información.



readFile y readFileSync

01

Lectura Asíncrona vs Síncrona

`readFile` permite lectura no bloqueante ideal para aplicaciones concurrentes, mientras `readFileSync` bloquea la ejecución, perfecto para configuraciones iniciales.

02

Manejo de Errores

En `readFile` se usan callbacks para errores, mientras `readFileSync` utiliza bloques try/catch más intuitivos para identificar problemas.

03

Consideraciones de Rendimiento

Elegir `readFile` para aplicaciones interactivas y `readFileSync` para scripts secuenciales según el contexto específico.

Estructura: `fs.readFile(path, options, callback)` `fs.readFileSync(path, options)`

writeFile y writeFileSync

Escritura Asíncrona

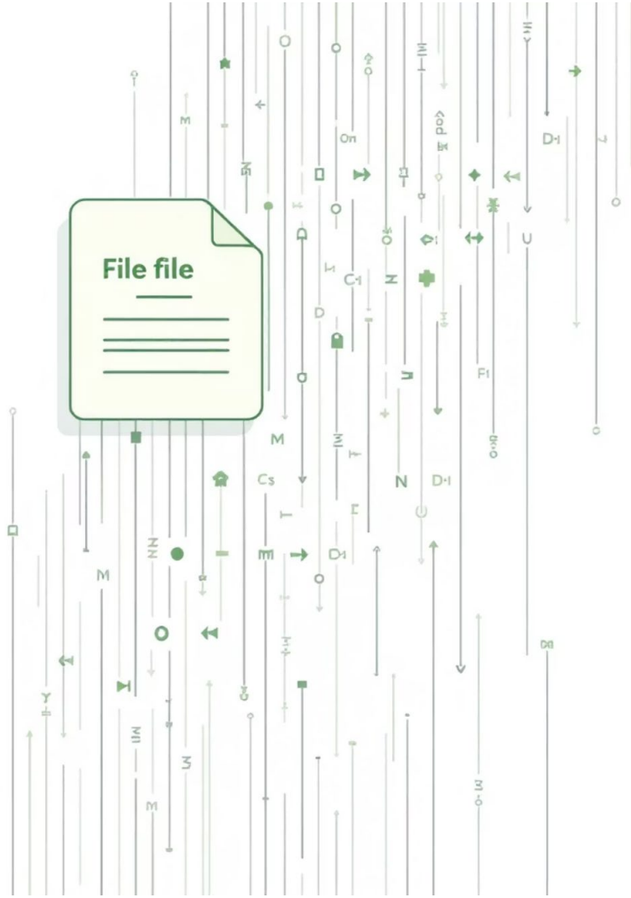
El método `writeFile` permite escritura no bloqueante, optimizando el rendimiento en aplicaciones con alta concurrencia y múltiples operaciones simultáneas.

Escritura Síncrona

`writeFileSync` garantiza finalización completa antes de continuar, esencial en scripts que requieren secuencialidad estricta.

Gestión de Errores Robusta

Implementar manejo adecuado de errores previene fallos críticos y asegura la integridad de los datos escritos.



Estructura: `fs.writeFile(path, data, options, callback)` `fs.writeFileSync(path, data, options)`

appendFile y

Funcionalidad Principal

Los métodos `appendFile` y `appendFileSync` permiten añadir contenido al final de un archivo existente sin sobrescribir el contenido previo.

Casos de Uso Comunes

Ideales para registros de logs, acumulación de datos, y situaciones donde necesitas preservar información histórica mientras añades nueva.

Estructura: `fs.appendFile(path, data, options, callback)`fs.appendFileSync(path, data, options)

unlink y unlinkSync

Eliminación de Archivos

Los métodos `unlink` y `unlinkSync` permiten eliminar archivos del sistema de forma definitiva, ofreciendo control total sobre la gestión de archivos temporales o innecesarios.

Precauciones Importantes

La eliminación es permanente e irreversible. Siempre verificar la existencia del archivo y implementar confirmaciones antes de ejecutar estas operaciones.

Estructura: `fs.unlink(path, callback)` `fs.unlinkSync(path)`

mkdir y mkdirSync

Creación de Directorios

Estos métodos permiten crear nuevos directorios en el sistema de archivos, siendo fundamentales para organizar la estructura de datos de tu aplicación.

Método Síncrono Ideal

`mkdirSync` es perfecto para scripts que necesitan asegurar la existencia de directorios antes de realizar otras operaciones críticas.

Estructura: `fs.mkdir(path, options, callback)`fs.mkdirSync(path, options)

readdir

Lectura de Contenido de Directorio

El método `readdir` permite examinar el contenido completo de un directorio, devolviendo una lista con todos los nombres de archivos y subdirectorios contenidos.

Exploración Dinámica

Especialmente útil para aplicaciones que necesitan procesar múltiples archivos, crear índices dinámicos o implementar funcionalidades de exploración de archivos.

Estructura: fs.readdir(path, [options], callback)

Buenas prácticas al trabajar con el módulo fs

Preferir Métodos Asíncronos

Utilizar funciones asíncronas previene bloqueos del hilo principal, manteniendo la aplicación responsiva y mejorando significativamente la experiencia del usuario.

Implementar Manejo de Errores

Verificar y gestionar errores en todas las operaciones de archivo es fundamental para mantener la estabilidad y confiabilidad de la aplicación.

Limpiar Recursos Adecuadamente

Eliminar archivos temporales y cerrar recursos correctamente previene pérdidas de memoria y optimiza el rendimiento general del sistema.