10 NodeJS: Módulos

Node.js ha evolucionado en su sistema de módulos, pasando de CommonJS a ES Modules (ECMAScript Modules). Te explico cómo funciona y por qué hay estos cambios:

CommonJS (el sistema antiguo)

Este fue el sistema original de Node.js y sigue siendo ampliamente usado:

```
// Importación
const fs = require("fs");

// Exportación
module.exports = suma;
// o
exports.suma = suma;
```

Características:

- Sincrónico (en el momento de cargar el módulo)
- `require()` es una función de Node
- `module.exports` y `exports` son objetos especiales

ES Modules (el sistema moderno)

Este es el sistema estándar de JavaScript (ECMAScript):

```
// Importación
import { readFile } from 'fs';

// Exportación
export const suma = (a, b) => a + b;
```

Características:

- Asincrónico (mejor para optimización)
- Sintaxis estándar de JavaScript
- Compatible con navegadores
- Soporte para importaciones dinámicas con `import()`

¿Por qué el cambio?

1. **Estandarización**: ES Modules es parte del estándar ECMAScript, mientras que CommonJS era específico de Node.js.
2. **Mejor rendimiento**: ES Modules permite análisis estático y tree-shaking (eliminar código no usado).

3. **Compatibilidad con navegadores**: Mismo sistema en frontend y backend.
4. **Características avanzadas**: Soporte para importaciones dinámicas y top-level await.

Configuración necesaria

Para usar ES Modules en Node.js tienes estas opciones:

1. Agregar en `package.json`:

```
{ "type": "module" }
```

1. O usar extensión `.mjs` para los módulos (sin necesidad de configurar `package.json`)

Situación actual (2025)

- **Node.js soporta ambos sistemas**, pero hay una transición hacia ES Modules.
- **CommonJS sigue siendo totalmente funcional** y se usa en muchos proyectos existentes.
- **Las diferencias principales** son sintácticas, pero también hay diferencias en:
  - Resolución de rutas
  - Comportamiento de `this`
  - Soporte para importaciones dinámicas
  - Caché de módulos

Recomendaciones

1. **Proyectos nuevos**: Usa ES Modules (configura `"type": "module"`).
2. **Proyectos existentes**: Puedes seguir con CommonJS o migrar gradualmente.
3. **Librerías**: Considera soportar ambos sistemas si es una librería pública.

La transición ha sido gradual para no romper la compatibilidad con el ecosistema existente de Node.js.