

FullStack Developer

Ecosistema de Node.js

Node.js

Node.js es un entorno de ejecución de JavaScript del lado del servidor que permite a los desarrolladores crear aplicaciones de red escalables y de alto rendimiento.

1. Node.js Core

Node.js está construido sobre el motor V8 de Google, que es el mismo motor que se usa en Google Chrome para ejecutar JavaScript. El núcleo de Node.js incluye bibliotecas que implementan las funciones básicas del sistema operativo, como la manipulación de archivos, la creación de servidores HTTP, y el manejo de streams y buffers.

2. NPM (Node Package Manager)

NPM es el gestor de paquetes predeterminado para Node.js y es una de las herramientas más importantes en el ecosistema de Node.js. Permite a los desarrolladores instalar, compartir y gestionar dependencias (módulos o paquetes) que pueden ser reutilizados en diferentes proyectos.

3. Módulos y Paquetes

Node.js adopta un sistema de módulos para gestionar y organizar el código. Los módulos pueden ser de tres tipos:

- **Módulos Nativos:** Incluidos en Node.js, como fs (file system), http, y path.
- **Módulos de Terceros:** Paquetes disponibles en el registro de NPM, como Express, Mongoose, y Lodash.
- **Módulos Locales:** Creados por los desarrolladores para estructurar sus propias aplicaciones.

4. Frameworks y Librerías Populares

Existen numerosos frameworks y librerías que se han construido sobre Node.js para facilitar el desarrollo de aplicaciones. Algunos de los más populares incluyen:

- **Express:** Un framework web minimalista y flexible que proporciona un conjunto robusto de características para aplicaciones web y móviles.
- **Koa:** Creado por los desarrolladores de Express, es más pequeño, más expresivo y robusto para el desarrollo de aplicaciones web.

- Socket.io:** Una librería para aplicaciones en tiempo real, permitiendo la comunicación bidireccional en tiempo real entre clientes y servidores.

- NestJS:** Un framework para construir aplicaciones eficientes y escalables del lado del servidor, basado en TypeScript.

5. Herramientas de Desarrollo

El ecosistema de Node.js incluye diversas herramientas para mejorar la productividad del desarrollador:

- Nodemon:** Monitorea los cambios en el código y reinicia automáticamente la aplicación.

- Webpack:** Un empacador de módulos que permite compilar archivos JavaScript.

- Babel:** Un compilador de JavaScript que permite usar la última sintaxis de JavaScript sin preocuparse por la compatibilidad con navegadores antiguos.

6. Servicios en la Nube y Plataforma como Servicio (PaaS)

Existen muchas plataformas que permiten desplegar aplicaciones Node.js fácilmente:

- Render:** Plataforma como servicio que permite desplegar, gestionar y escalar aplicaciones.
- Vercel:** Ideal para despliegues rápidos de aplicaciones web front-end y back-end.
- AWS (Amazon Web Services):** Ofrece servicios como AWS Lambda, EC2, y Elastic Beanstalk para desplegar aplicaciones Node.js.

7. Bases de Datos

Node.js se puede integrar con diversas bases de datos. Algunas comunes incluyen:

- MongoDB:** Una base de datos NoSQL de documentos que se integra bien con Node.js mediante Mongoose.

•**PostgreSQL y MySQL:** Bases de datos SQL que se pueden manejar mediante librerías como pg para PostgreSQL y mysql2 para MySQL.

•**Redis:** Una base de datos en memoria utilizada como caché y almacenamiento de datos en tiempo real.

Nodemon.js

Nodemon es una herramienta que se utiliza en el desarrollo de aplicaciones Node.js para facilitar el proceso de desarrollo. Su función principal es monitorear los archivos de tu proyecto y reiniciar automáticamente el servidor cada vez que se detectan cambios en el código fuente. Esto elimina la necesidad de detener y reiniciar manualmente el servidor cada vez que realizas una modificación, lo cual mejora significativamente la productividad del desarrollador.

Características Principales de Nodemon

1.Reinicio Automático: Nodemon monitorea los cambios en los archivos de tu proyecto y reinicia automáticamente la aplicación cuando se detecta un cambio.

2.Configuración Personalizable: Puedes configurar Nodemon para que observe archivos específicos, extensiones de archivos y directorios. También puedes ignorar archivos o directorios específicos.

3.Compatibilidad con Lenguajes de Transpilación: Nodemon puede ser utilizado con lenguajes que requieren transpilación, como TypeScript, Babel, CoffeeScript, etc., permitiendo configuraciones adicionales para ejecutar comandos de transpilación antes de reiniciar el servidor.

- 4. Integración con Herramientas de Desarrollo:** Nodemon se integra fácilmente con otras herramientas de desarrollo y scripts npm, facilitando un flujo de trabajo más eficiente.
- 5. Uso Interactivo:** Nodemon proporciona una CLI interactiva que permite detener o reiniciar manualmente el servidor con comandos sencillos mientras la aplicación está en ejecución.

Instalación de Nodemon

Podemos instalar Nodemon globalmente o como una dependencia de desarrollo en el proyecto. Abrimos la consola (o terminal) de Visual Studio code y recomiendo instalar de manera global.

•Instalación global:

```
npm install -g nodemon
```

•Instalación como dependencia de desarrollo:

```
npm install --save-dev nodemon
```


Uso Básico

Una vez instalado, puedes usar Nodemon para ejecutar tu aplicación en lugar de node. Por ejemplo, si normalmente ejecutamos una aplicación con:

```
node app.js
```

Con Nodemon, usaríamos:

```
nodemon app.js
```

Ejemplo de Uso en un Proyecto

```
project/  
├─ src/  
│   └─ app.js  
└─ package.json
```

Para usar Nodemon en este proyecto:

1- Si lo tenemos instalado globalmente (lo recomendado), abrimos la consola y colocamos

```
npm init -y
```

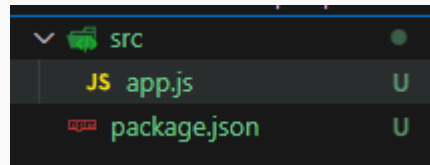
Obtenemos este json que vamos a completar y modificar para nuestro proyecto

```
{  
  "name": "code",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC"  
}
```

2- Deberá quedar así por el momento

```
{
  "name": "code",
  "version": "1.0.0",
  "description": "",
  "type": "module",
  "main": "app.js",
  "scripts": {
    "dev": "nodemon ./src/app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

- type: en "module" para poder trabajar con ES6
- main: es el punto de entrada o entrypoint app.js
- scripts: con "dev" vamos a ejecutar nodemon al módulo de js llamado app.js que está dentro de la carpeta "src"



Vamos a empezar a escribir código en app.js, un `console.log("hola mundo")` para empezar, abrimos la consola y colocamos

`npm run dev`

```
JS app.js U X
src > JS app.js
1 console.log('hola mundo')
```

```
> code@1.0.0 dev
> nodemon ./src/app.js

[nodemon] 3.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node ./src/app.js`
hola mundo
[nodemon] clean exit - waiting for changes before restart
█
```

Y acá vemos como nuestra app empezó a funcionar con nodemon

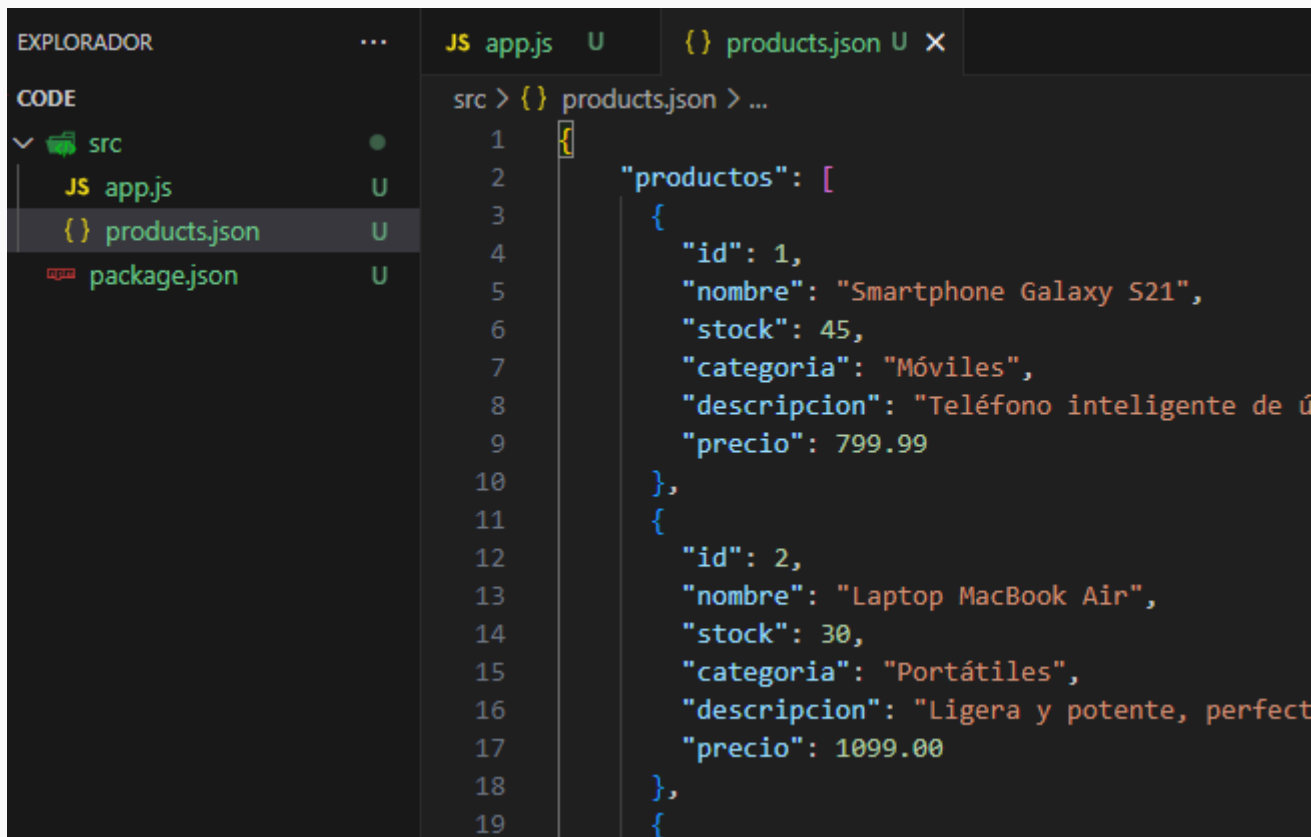
Como ya vimos, esto nos va a permitir poder modificar nuestro proyecto sin la necesidad de parar el proceso e ir viendo los cambios en tiempo real.

Instalemos algunas dependencias para interactuar un poco más y hacer una app que se pueda ejecutar desde la consola (terminal)

Vamos a usar fs (File System) este nos permite leer archivos como json, txt y otros
cli-table3 nos va a crear una tabla en la consola
y por último chalk que nos va a permitir darle colores a nuestros datos

```
npm install cli-table3 chalk
```

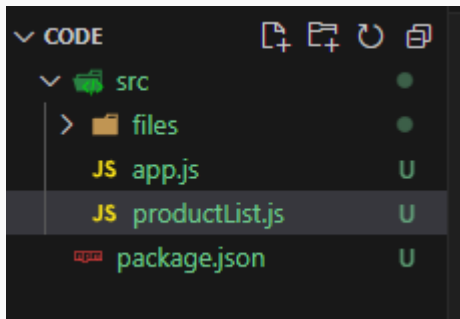
Tenemos un json de productos, este almacena un objeto que tiene una key productos y esta tiene un array de objetos



```
EXPLORADOR  ...  JS app.js  U  {} products.json  U  X
CODE
src
  JS app.js  U
  {} products.json  U
  package.json  U

src > {} products.json > ...
1  {
2    "productos": [
3      {
4        "id": 1,
5        "nombre": "Smartphone Galaxy S21",
6        "stock": 45,
7        "categoria": "Móviles",
8        "descripcion": "Teléfono inteligente de última generación",
9        "precio": 799.99
10     },
11     {
12       "id": 2,
13       "nombre": "Laptop MacBook Air",
14       "stock": 30,
15       "categoria": "Portátiles",
16       "descripcion": "Ligera y potente, perfecta para trabajo y estudio",
17       "precio": 1099.00
18     },
19     {
```

Vamos a empezar a modularizar, vamos a crear una carpeta "files" para guardar nuestro json y además un módulo de js que se encargará de leer ese json



En productList.js

```
import fs from 'fs';

const data = fs.readFileSync('./src/files/products.json', 'utf8');

export default data
```

ReadFileSync es una función de Node.js que lee el contenido de un archivo de forma síncrona. Sus características principales son:

1. Lee todo el contenido del archivo de una vez.
2. Es una operación bloqueante, lo que significa que detiene la ejecución del programa hasta que se complete la lectura.
3. Devuelve el contenido del archivo como un buffer o una cadena, dependiendo de las opciones especificadas.

4. Útil para leer archivos de configuración o datos que se necesitan antes de que el programa pueda continuar.
5. No recomendada para archivos grandes o en operaciones donde el rendimiento es crítico, ya que puede ralentizar la aplicación.
6. Requiere la ruta del archivo como argumento.
7. Puede lanzar excepciones si hay problemas al acceder al archivo.

Como este método devuelve una cadena debemos parsearla con `JSON.parse` así obtendremos un tipo de estructura que js pueda interpretar de manera correcta

Sin el parseo

```
{
  "productos": [
    {
      "id": 1,
      "nombre": "Smartphone Galaxy S21",
      "stock": 45,
      "categoria": "Móviles",
      "descripcion": "Teléfono inteligente de última generación con cámara de alta resolución",
      "precio": 799.99
    },
    {
      "id": 2,
      "nombre": "Laptop MacBook Air",
      "stock": 30,
      "categoria": "Portátiles",
      "descripcion": "Ligera y potente, perfecta para trabajo y entretenimiento",
      "precio": 1099.00
    }
  ]
}
```


Y modificando un poco el código

```
import fs from 'fs';

const data = JSON.parse(fs.readFileSync('./src/files/products.json', 'utf8'));

export default data
```

```
{
  productos: [
    {
      id: 1,
      nombre: 'Smartphone Galaxy S21',
      stock: 45,
      categoria: 'Móviles',
      descripcion: 'Teléfono inteligente de última generación con cámara de alta resolución',
      precio: 799.99
    },
    {
      id: 2,
      nombre: 'Laptop MacBook Air',
      stock: 30,
      categoria: 'Portátiles',
      descripcion: 'Ligera y potente, perfecta para trabajo y entretenimiento',
      precio: 1099
    }
  ]
}
```

La consola ya nos indica otra cosa, incluso para repasar podríamos usar `typeof` en un `console.log` para ver el tipo de datos que nos devuelve en ambas ocasiones, dando como resultado que para poder trabajar deberíamos usar `JSON.parse`.

Ahora vamos a crear la tabla con `cli-table3` y `chalk` para darle color, creamos el módulo `table.js`


```
import Table from "cli-table3";
import chalk from "chalk";

const table = new Table({
  head: [
    chalk.green('id'),
    chalk.green('nombre'),
    chalk.green('stock'),
    chalk.green('precio'),
    chalk.green('categoria'),
    chalk.green('descripcion')
  ],
  colWidths: [10, 20, 10, 10, 20, 30]
});

export default table
```

Tenemos un objeto con 2 keys, una que representa la cabecera de la tabla, esta es head que tiene como valor un array de strings, que se pasan como argumento a chalk.green para tomar un color a la hora de mostrar por consola, chalk tiene muchos colores para mostrar, lo veremos a la hora de mostrar los datos.

La key colWidths define el ancho de cada columna, y los elementos del array van en correspondencia con el array de head.



```
JS app.js U X JS table.js U JS productList.js U
src > JS app.js > ...
1 import data from './productList.js';
2 import table from './table.js';
3
4 console.log(table.toString())
5
```

En app.js llamamos a table y para poder visualizar usamos el método toString()

Ejecutamos "npm run dev"

```
> code@1.0.0 dev  
> nodemon ./src/app.js
```

```
[nodemon] 3.0.2  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node ./src/app.js`
```

id	nombre	stock	precio	categoria	descripcion
----	--------	-------	--------	-----------	-------------

```
[nodemon] clean exit - waiting for changes before restart
```

Ahora vamos a implementar datos pero con una función que se va a encargar de poner en rojo a aquellos stocks que sean menores a cierto número.

```
function checkStock(stock){  
    return stock < 30 ? chalk.bgRed.white(stock) : chalk.green(stock);  
}
```

Esta función retorna un string con los estilos según corresponda

Ahora recordemos como tenemos los datos en el archivo json, cuando parseamos esto tenemos un objeto con una key "productos" y este un value que es el array de objetos productos.

```
{  
  "productos": [  
    {  
      "id": 1,  
      "nombre": "Smartphone Galaxy S21",  
      "stock": 45,  
      "categoria": "Móviles",  
      "descripcion": "Teléfono inteligente",  
      "precio": 799.99  
    },  
    {  
      "id": 2,  
      "nombre": "Smartphone Galaxy S21",  
      "stock": 45,  
      "categoria": "Móviles",  
      "descripcion": "Teléfono inteligente",  
      "precio": 799.99  
    }  
  ]  
}
```

Lo que vamos a hacer es con un map convertir esos datos para luego implementarlos en la table

```
table.push(  
  [chalk.yellow('John Doe'), chalk.cyan('30'), chalk.magenta('Software Developer')],  
  [chalk.yellow('Jane Smith'), chalk.cyan('28'), chalk.magenta('Graphic Designer')],  
  [chalk.yellow('Sam Johnson'), chalk.cyan('35'), chalk.magenta('Product Manager')]  
);
```

Ya que el método push necesita recibir arrays como argumentos, cada array será una fila

```
const checkProducts = data.productos.map(product => {  
  return [  
    chalk.whiteBright(product.id),  
    product.nombre,  
    checkStock(product.stock),  
    product.precio,  
    product.categoria,  
    product.descripcion  
  ]  
})
```

Ahora ya tenemos un array de arrays

Para hacer un push usaremos un spread operator

```
table.push(...checkProducts)
```

Luego...

```
console.log(table.toString())
```

Este debería ser el resultado de nuestro app.js

```
import data from "./productList.js";
import table from "./table.js";
import chalk from "chalk";

function checkStock(stock){
    return stock < 30 ? chalk.bgRed.white(stock) :
    chalk.green(stock);
}

const checkProducts = data.productos.map(product => {
    return [
        chalk.whiteBright(product.id),
        product.nombre,
        checkStock(product.stock),
        product.precio,
        product.categoria,
        product.descripcion
    ]
})

table.push(...checkProducts)

console.log(table.toString())
```


Resultado

```
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node ./src/app.js`
```

id	nombre	stock	precio	categoria	descripcion
1	Smartphone Galaxy...	45	799.99	Móviles	Teléfono inteligente de últ...
2	Laptop MacBook Air	30	1099	Portátiles	Ligera y potente, perfecta ...
3	Auriculares Inalá...	60	349.99	Audio	Auriculares con cancelación...
4	Tablet iPad Pro	25	799	Tablets	Potente tablet para profes...
5	Smart TV LG OLED ...	15	1499.99	Televisores	Televisor inteligente con c...
6	Cámara Mirrorless...	20	1999.99	Fotografía	Cámara profesional sin espe...
7	Consola PlayStati...	10	499.99	Videojuegos	La última consola de Sony c...

Vemos como el stock me marca en rojo a aquellos productos que son menores a 30

Ejercicios

Pueden optar por 2 caminos

- Hacer una tabla similar a esta con los datos que quieran
- Implementar una librería de Node para realizar algún ejercicio similar, es decir, implementando nodemon y el manejo de dependencia por medio de package.json

Cualquiera de los 2 estará perfecto.