

Clase 13: Introducción a Node

Sitio:	Centro de E-Learning - UTN.BA	Imprimido	Nilo Crespi
Curso:	Curso de Backend Developer - Turno Noche	por:	
Libro:	Clase 13: Introducción a Node	Día:	Friday, 23 de January de 2026, 10:25

Tabla de contenidos

[**1. Introducción**](#)

[**2. Introducción a Node**](#)

[**3. EXPRESS**](#)

1. Introducción

Node.js es un entorno de ejecución de JavaScript basado en el motor V8 de Chrome, diseñado para desarrollar aplicaciones del lado del servidor de manera eficiente. Su modelo de entrada/salida asíncrono y orientado a eventos lo hace ideal para la creación de aplicaciones escalables y de alto rendimiento. Además, cuenta con un extenso ecosistema de paquetes gestionado a través de npm, lo que permite acceder a una gran variedad de herramientas y librerías para optimizar el desarrollo.

El uso de Node.js es fundamental en el desarrollo moderno, ya que permite programar tanto el frontend como el backend utilizando un solo lenguaje: JavaScript. Esto facilita la integración entre ambas partes de una aplicación y agiliza los procesos de desarrollo. Su flexibilidad y el soporte de una amplia comunidad lo han convertido en una opción clave para construir aplicaciones web, servidores, APIs y servicios en la nube.

2. Introducción a Node

Node.js

Node.js es un entorno de ejecución para construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S (Entrada/Salida) sin bloqueo y orientado a eventos, que lo hace liviano y eficiente.

El ecosistema de paquetes de Node.js, npm, es el ecosistema más grande de librerías de código abierto en el mundo.

¿Por qué es importante conocerlo?

Es una herramienta muy valiosa, ya que les permite crear aplicaciones del lado del servidor de manera rápida y eficiente utilizando JavaScript. Además, Node.js cuenta con una amplia comunidad de desarrolladores y una gran cantidad de paquetes disponibles a través de npm, lo que facilita el desarrollo de aplicaciones complejas.

- Fue concebido como un entorno de ejecución de JavaScript orientado a eventos asíncronos y está diseñado para construir aplicaciones en red escalables.
- Node permite además la creación de sitios y aplicaciones web enteramente con JavaScript, así como también la creación y utilización de un sin fin de herramientas.

Instalación



Sitio oficial <https://nodejs.org/es/>.

- Una vez terminada la instalación podemos verificar que todo haya salido correctamente abriendo la consola (CMD) y escribiendo **node -v**.

Uso básico de Node

En su forma más simple, podemos abrir la consola y escribir solamente **node**. Esto abrirá una instancia que nos permitirá ir ingresando expresiones de Javascript que serán evaluadas e impresas en el momento

Algunos ejemplos de cosas que podemos probar:

- Operaciones matemáticas
- Asignaciones de variables
- Definición de funciones
- Y cualquier otro código Javascript válido

Para salir de la instancia de node escribimos **.exit** (con punto antes de la e) o presionamos la combinación de teclas CTRL+C.

También podemos ejecutar scripts directamente con Node. Creamos un archivo nuevo llamado index.js y escribimos el siguiente código:

```
let hoy = new Date()
console.log('Hoy es ' +
hoy)
for(i = 0; i < 10; i++)
{ console.log(i)
}
```

Lo guardamos y lo ejecutamos escribiendo **node app.js**.

El resultado debería ser la fecha y hora actual y un conteo de 0 a 9 como respuesta en la consola.

Inicialización de un proyecto con Node.js

- Para comenzar un proyecto con Node.js desde cero es necesario correr el script de inicialización **npm**

init. Este comando debe ser corrido desde la consola, en la carpeta donde deseemos inicializar el proyecto.

- El script nos hará una serie de preguntas básicas sobre nuestros proyecto. Podemos contestar todas apretando la tecla ENTER o llenar los datos que creamos necesarios.
- Si no deseamos completar los datos en ese momento, podemos utilizar **npm init -y**

package.json

Al finalizar, encontraremos un archivo llamado package.json donde se guardó toda la información que acabamos de cargar. Así mismo, en ese archivo, es donde el manejador de paquetes de Node, npm, va a ir guardando las dependencias o librerías de nuestro proyecto, así como también, varias de las herramientas que usaremos para desarrollo.

Instalación y uso de librerías con npm

La instalación de librerías se hace mediante el comando

npm install. A este debemos indicarle qué librería deseamos que baje e instale y este lo hará de forma automática. Las librerías las podemos buscar en el sitio <https://www.npmjs.com/>, que es el repositorio central de npm.



Ejemplo

A modo de ejemplo, vamos a instalar una librería llamada moment (<http://momentjs.com/>). Su página de npm es <https://www.npmjs.com/package/moment>, donde encontraremos información de la misma.

Ejecutamos el siguiente comando **npm install moment**.

Una vez terminada la instalación podemos abrir el archivo package.json para verificarlo.

package.json npm install.

En el apartado dependencias del archivo package.json se lista las librerías que requiere nuestro proyecto para funcionar. Las librerías que hayamos descargado de esta forma se graban automáticamente en la carpeta node_modules. Por convención, si estamos usando GIT se evita versionar esta carpeta agregándola al archivo .gitignore.

En caso de que trabajemos en un proyecto que incluya un archivo package.json podemos descargar todas sus dependencias utilizando el comando **npm install**.

package-lock.json

En la versión 5, npm introdujo el package-lock.json.

El objetivo del package-lock.json es realizar un seguimiento de la versión exacta de cada paquete que se instala para que un producto sea 100% reproducible de la misma manera.

Módulos y paquetes en Node.js

Los módulos se utilizan para encapsular y reutilizar el código. Crear un módulo utilizando la función "module.exports" y luego importarlo en otro archivo utilizando la función "require". Por ejemplo, el siguiente código crea un módulo que exporta una función que imprime un saludo en la consola:

```
// archivo saludo.js module.exports =
function(nombre) { console.log(`Hola, ${nombre}!`); }
// archivo app.js const saludo =
require('./saludo'); saludo('Juan'); // imprime
"Hello, Juan!" en la consola
```

File System

Documentación → <https://nodejs.org/dist/latest-v18.x/docs/api/fs.html>

Creación de un servidor HTTP básico

Node.js incluye el módulo "http" que permite crear servidores HTTP. Este código crea un servidor que responde con el mensaje "¡Hola, mundo!" a todas las solicitudes. El servidor se inicia en el puerto 3000 y se muestra un console log.

```
const http = require('http'); const server =
http.createServer((req, res) => {
res.write('¡Hola, mundo!'); res.end(); });
server.listen(3000, () => { console.log('Servidor
iniciado en el puerto 3000');
});
```

1. **Instalar node js.**
2. **Ejecutar código js con node.**
 1. **Crea un proyecto node con npm init.**
 2. **Instalá una dependencia npm npm install.**
 3. **Verifica tus archivos package.json y package-lock.json.**

3. EXPRESS

Express

Es el framework web más popular de Node, y es la librería subyacente para un gran número de otros frameworks de Node.



Proporciona mecanismos para:

- Escritura de manejadores de peticiones con diferentes verbos HTTP en diferentes caminos URL (rutas).
- Integración con motores de renderización de "vistas" para generar respuestas mediante la introducción de datos en plantillas.
- Añadir procesamiento de peticiones "middleware" adicional en cualquier punto dentro de la ejecución de una petición.

Nodemon

Es una herramienta que monitorea los cambios en el código fuente que se está desarrollando y automáticamente reinicia el servidor.

Es una herramienta muy útil para desarrollo de aplicaciones en node.js



Instalación:

<https://www.npmjs.com/package/nodemon>

Iniciar un proyecto

1. Ejecutar: `npm init -y`
2. Instalar Express con: `npm i express` <https://www.npmjs.com/package/express>
3. Creamos punto de acceso, por ejemplo, `app.js`
4. Ir a : `localhost:3000`

Rutas

En sitios web o aplicaciones web dinámicas, que accedan a bases de datos, el servidor espera recibir peticiones HTTP del navegador (o cliente).

Cuando se recibe una petición, la aplicación determina cuál es la acción adecuada correspondiente, de acuerdo a la estructura de la URL y a la información (opcional) indicada en la petición con los métodos POST o GET.

Dependiendo de la acción a realizar, puede que se necesite leer o escribir en la base de datos, o realizar otras acciones necesarias para atender la petición correctamente.

Métodos

Express posee métodos para especificar qué función ha de ser llamada dependiendo del verbo HTTP usado en la petición (GET, POST, DELETE, etc.) y la estructura de la URL ("ruta").

También tiene los métodos para especificar qué plantilla ("view") o gestor de visualización utilizar, donde están guardadas las plantillas de HTML que han de usarse y cómo generar la visualización adecuada para cada caso.

Definición

La definición de ruta tiene la siguiente estructura:

app.METHOD(PATH, HANDLER)

Donde:

- app es una instancia de express.
- METHOD es un método de solicitud HTTP.
- PATH es una vía de acceso en el servidor.
- HANDLER es la función que se ejecuta cuando se correlaciona la ruta.

Rutas

Los siguientes ejemplos ilustran las definiciones de rutas simples.

```
const express = require("express");
const app = express();
app.get("/", (req, res) => {
  res.send("Hola Mundo");
});
app.get("/contacto", (req, res) => {
  res.send("Contacto");
});
```