

DESARROLLO DE SERVICIOS WEB CON NODEJS



Email: *julietalanciotti@gmail.com*

Módulo 2 - NodeJS y Express

Creando un servidor web básico

Paso a Paso

Desventajas

Express

Instalación

EndPoint web

Creando un EndPoint

Primer paso

Segundo paso

Rutas

Rutas con parámetros y body

Anexo

Archivos JSON

Convenciones utilizadas por JSON

¿Para qué se utiliza?

Los datos en un archivo JSON

Crear un objeto JSON en JavaScript

Creando un servidor web básico

Como ya vimos, **Node** es un entorno de código abierto, que trabaja en tiempo de ejecución. Es multiplataforma y permite a los desarrolladores crear toda clase de herramientas de lado del servidor y aplicaciones en JavaScript. Ya que la ejecución en tiempo real está pensada para utilizarse fuera del contexto de un explorador, es posible crear nuestro propio servidor web de manera sencilla.

Node nos permite utilizar el paquete HTTP para poder crear un servidor que escuchará cualquier petición en la URL <http://127.0.0.1:5000/>

Paso a Paso

 Link a video explicado paso a paso: <https://youtu.be/2jaWjNCwurM>

1. Debemos crear un proyecto. Para esto utilizamos el comando `npm init` Nos pedirá una serie de datos para la configuración. Podemos presionar enter para dejar los datos por defecto.
2. Una vez creado el proyecto, se habrá generado un archivo `package.json`. Este archivo guarda la configuración de nuestro proyecto:

```
//ARCHIVO PACKAGE.JSON

{
  "name": "ejemplo_2",
  "version": "1.0.0",
  "description": "ejemplo basico con express y haciendo un get",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.1"
  }
}
```

Notar que en la parte de **“scripts”** es importante indicar qué archivo se ejecutará primero. Es por eso que debemos indicarle en la sentencia **“start”** que ejecutaremos el archivo **index.js** bajo NODE.

3. Luego creamos nuestro archivo **index.js**, el cuál tendrá el código básico de un servidor HTTP para una prueba sencilla.

```
// cargamos el módulo HTTP
var http = require("http");

// creo el servidor HTTP y hago que escuche el puerto 5000
http.createServer(function(request, response) {

    // defino la cabecera HTTP, con el estado 200 (ok) y el tipo de contenido
    response.writeHead(200, {'Content-Type': 'text/plain'});

    // mensaje de respuesta
    response.end('Hola Mundo!');

}).listen(5000);

// escribo un mensaje para indicar en que ruta se encuentra el servidor
console.log('Servidor en la url http://127.0.0.1:5000/');
```

4. Por último, solo falta correr nuestro servidor, para esto utilizaremos el comando `npm start`



Si querés probar el código en tu computadora, te dejo el link a GitHub:

https://github.com/JL-Curso-NodeJS/ejemplos-basicos/blob/main/ejemplo_1/index.js

Cabe aclarar que esto podremos hacerlo una vez que tengamos instalado Node.js y NPM en nuestra computadora. Para entender un poco más cómo hacer esto, te recomiendo que mires el video: <https://youtu.be/CqjXtmUyww4>

Desventajas

1. Algunas tareas comunes de desarrollo web no están directamente soportadas por el mismo **Node**.
 2. Si se desea añadir el manejo específico de verbos HTTP (GET, POST, PUT, etc), se debe escribir todo el código.
 3. Si se desea gestionar de forma separada las peticiones por medio de diferentes rutas, tampoco se podría realizar de una manera sencilla y ordenada.
-


Express

Para solucionar los problemas anteriormente mencionados, y facilitarnos la vida a la hora de programar, se desarrolló un framework llamado **Express**.

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

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.
- Establecer ajustes de aplicaciones web como por ejemplo, qué puerto usar para conectar, y la localización de las plantillas que se utilizar para renderizar la respuesta.
- Añadir procesamiento de peticiones “*middleware*” adicional en cualquier punto de la petición.

 Lista de paquetes middleware mantenida por el equipo de Express Middleware
<https://expressjs.com/es/resources/middleware.html>

Instalación

1. Dentro de nuestro proyecto deberemos escribir el comando:

```
npm install express --save
```

2. Con la instalación de Express, se habrán agregado varios archivos a nuestro proyecto:
 1. **directorio node_modules:** guarda todas las dependencias y paquetes que tendrá instalado nuestro proyecto.
 2. **archivo package.json:** se habrá agregado como dependencia la versión de express.
 3. **archivo package-lock.json:** sencillamente evita este comportamiento general de actualizar versiones minor o fix de modo que cuando alguien clona nuestro repositorio y ejecuta npm install en su equipo, npm examinará package-lock.json e instalará la versión exacta de los paquete que nosotros habíamos instalado, ignorando así los ^ y ~ de package.json.

EndPoint web

Un **endpoint** es una URL de una API o un backend que responde a una petición. Los endpoints no están pensados para interactuar con el usuario final. Usualmente sólo devolverán un JSON, o no devolverán nada.

Adicionalmente, se asume que cuando se habla de un endpoint estamos en un entorno *RESTful*, por lo cual, el cliente puede usar un mismo endpoint con distintos verbos.

/usuarios

GET: Devolverá una lista de usuarios

POST: Creará un usuario

Creando un EndPoint



Link a video explicado paso a paso: <https://youtu.be/j8megukdhEs>

Primer paso

1. Instalar NodeJS
2. Instalar NPM
3. Instalar Express

Segundo paso

```
// importo express
var express = require('express');

// instancio objeto
var app = express();

// creo ruta del endpoint. En este caso sera GET
app.get('/', function (req, res) {
  res.send("Hola Mundo!");
})

// servidor escuchando en el puerto 3000
app.listen(3000, function() {
  console.log("Aplicacion de ejemplo, escuchando el puerto 3000");
})
```



Si querés probar el código en tu computadora, te dejo el link a GitHub

https://github.com/JL-Curso-NodeJS/ejemplos-basicos/blob/main/ejemplo_2/index.js

Rutas

Uno de los elementos más importantes de la arquitectura REST son los recursos. Estos elementos de información son identificados por una URI (Identificador Único del Recurso), que deben presentar las siguientes características:

- Las URI recibirán nombres que no deben implicar una acción, es decir, se debe evitar colocar verbos en ellas.
 - <http://api.com/usuarios/12/borrar> INCORRECTO
 - <http://api.com/usuarios/12> CORRECTO
- Deben ser únicas, no debemos tener más de una URI para identificar un mismo recurso.
- Deben ser independientes de formato, es decir, no debe representar ninguna extensión.
- Deben mantener una jerarquía lógica. Es el criterio por el que se ordenan los elementos.
 - <http://api.com/fotos/12/usuario/224> INCORRECTO
 - <http://api.com/usuarios/224/fotos/12> CORRECTO
- Los filtrados de información de un recurso no se hacen en la URI
 - <http://mi-api.com/fotos/fecha/octubre> INCORRECTO
 - <http://mi-api.com/fotos?fecha=octubre> CORRECTO

```
// ruta para GET
app.get('/', function (req, res) {
    res.send('GET request to the homepage');
});

// ruta para POST
app.post('/', function (req, res) {
    res.send('POST request to the homepage');
});
```

Rutas con parámetros y body

En caso de que nuestros endpoints esperen recibir algún tipo de información, hay varias formas de recibirla. Pero veremos dos casos:

- **Params**

Utilizado por ejemplo cuando se hace un GET.

```
// req: request, res: response
app.get('/:nombre', (req, res) => {

    // envia una respuesta y extrae el nombre del parámetro
    res.send('Hola ' + req.params.nombre);

});
```

En este ejemplo, se está recibiendo por parámetro un string. Para acceder a este dato utilizamos `req.params` y el nombre del parámetro, en este caso

nombre

- **Body**

Utilizado por ejemplo cuando se hace un POST

```
// req: request, res: response
app.post('/', (req, res) => {

  // envia una respuesta y extraigo el nombre del body
  res.send('Hola ' + req.body.nombre);

});
```

En este ejemplo, se está recibiendo la información a través del **body** (por medio de un JSON). Para acceder a este, necesitamos desarmarlo y acceder a los datos. En nuestro caso, el json solo estaba compuesto de un campo llamado **nombre**, es por eso que utilizamos **req.body.nombre**

Para utilizar esto, deberemos escribir la siguiente configuración:

- **app.use(express.json())** Esto permite que Express entienda JSON
- **app.use(express.urlencoded({extended:true}))** ;

Permite que express entienda formularios enviados por post

 **Podés ver un ejemplo de esto:**

https://github.com/JL-Curso-NodeJS/ejemplos-basicos/tree/main/ejemplo_3

Anexo

Archivos JSON

JSON (notación de objetos javascript) es un formato de intercambio de datos. Aunque muchos lenguajes de programación lo soportan, JSON es especialmente útil al escribir cualquier tipo de aplicación basada en JavaScript, incluyendo sitios web y extensiones del navegador. Por ejemplo, es posible almacenar la información del usuario en formato JSON en una cookie o almacenar las preferencias de extensión en JSON en una cadena de valores de preferencias del navegador.

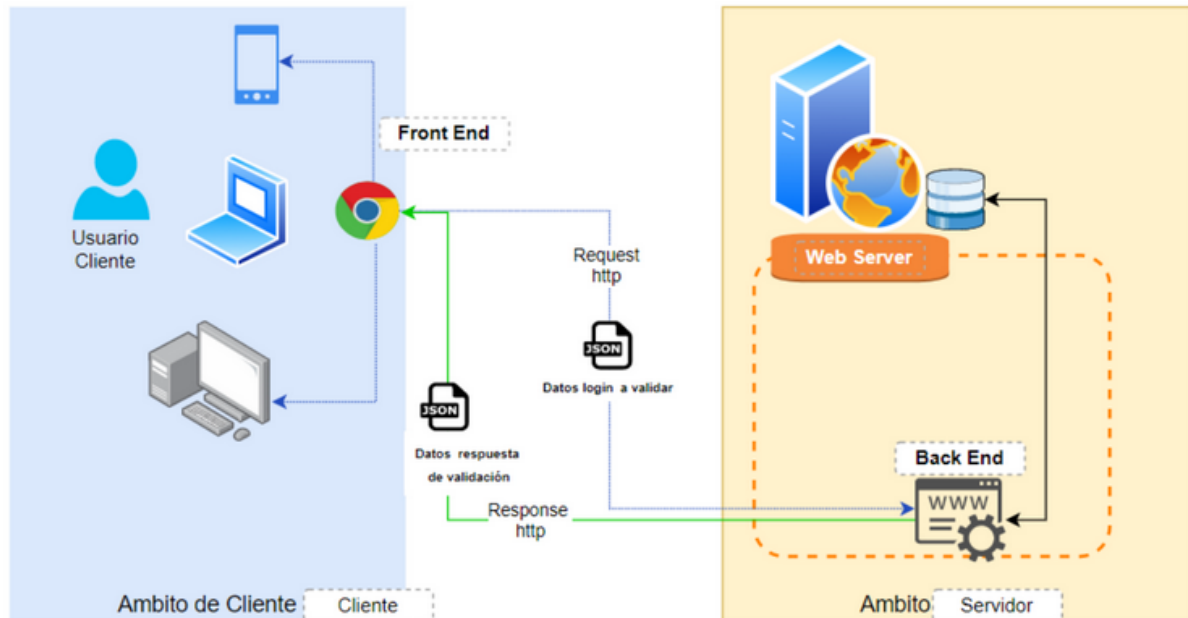
JSON es capaz de representar números, valores lógicos, cadenas, valores nulos, arreglos y matrices (secuencias ordenadas de valores) y objetos compuestos de estos valores. No representa de manera nativa tipos de datos más complejos como funciones, expresiones regulares, fechas, etc.

Convenciones utilizadas por JSON

- JSON son las siglas de JavaScript Object Notation.
- El formato fue especificado por Douglas Crockford.
- Fue diseñado para el intercambio de datos legibles por humanos.
- Se ha ampliado desde el lenguaje de secuencias de comandos JavaScript.
- La extensión del nombre de archivo es **.json**
- El tipo de Internet Media type es application / json

¿Para qué se utiliza?

1. Para serializar y transmitir datos estructurados a través de una conexión de red.
2. Para transmitir datos entre un servidor y aplicaciones web.
3. Los servicios web y las API lo utilizan para proporcionar datos vía internet.
4. La mayoría de las bases de datos pueden almacenar archivos JSON.



1. *Usuario ingresa sus datos en el front.*
2. *El código del front toma los datos y los serializa en formato JSON y los envía al back.*
3. *El back recibe el JSON y lo deserializa para ser procesados.*
4. *Luego del procesamiento, el back serializa la respuesta y la envía al front.*
5. *El front deserializa el JSON y muestra el resultado al usuario.*

Los datos en un archivo JSON

Los datos se escriben como pares de nombre/valor, al igual que las propiedades de los objetos de JavaScript. Un par de nombre/valor consta de un nombre de campo (entre comillas dobles), seguido de dos puntos, seguido de un valor:

```
alumno: {  
  
  "nombre": "Julieta",  
  "apellido": "Lanciotti",  
  "edad": 26,  
  "localidad": "La Plata"  
  
}
```

Crear un objeto JSON en JavaScript

- *Objeto vacío*

```
var JSONobj = {};
```

- *Creación de un nuevo objeto*

```
var JSONobj = new Object();
```

- *Creación de un objeto con atributo **nombre_persona** con una cadena de caracteres como valor, y un atributo **edad** con un valor numérico. Para acceder al atributo se utiliza el punto “.”*

```
var JSONobj = {  
  "nombre": "Juan",  
  "edad": 51  
}
```