



Buenos
Aires
Ciudad
Agencia de Habilidades
para el Futuro

<Talento
Tech/>

Node.JS

Clase N° 13: Modelo de datos y trabajo con JSON

¡Les damos la bienvenida!

 Vamos a comenzar a grabar la clase.

Índice



1



2

Modelo de datos y trabajo con JSON

- **JSON**
 - Parse
 - Stringify
- **Filesystem**
- **Modelos**

Datos en la nube

- **Firebase**
- **Firestore**
 - Nuevo Proyecto
 - Creando una colección
 - Configuración de Firebase Firestore en tu Proyecto

Objetivos de la Clase

1

Comprender Modelos

Aprenderemos a definir y utilizar modelos para organizar datos en una API Rest.

2

Dominar JSON

Exploraremos cómo convertir objetos JavaScript a JSON y viceversa.

3

Manejar Filesystem

Aprenderemos a acceder y manipular datos almacenados en archivos.

JSON

Introducción a JSON

JSON es un formato de texto ligero para intercambio de datos. Es fácil de leer y escribir para humanos y máquinas.

Se usa ampliamente en aplicaciones web, especialmente en APIs REST. Surgió como alternativa a XML.

Ejemplo de JSON

```
{  
  "name": "Juan",  
  "age": 30,  
  "isStudent": false,  
  "courses": ["Matemáticas",  
  "Programación"]  
}
```

Para realizar estas conversiones, se utilizan los métodos `JSON.parse()` y `JSON.stringify()`, los cuales permiten transformar datos de JSON a objetos de JavaScript y viceversa, facilitando así el intercambio de información en aplicaciones web.

JSON.parse()

Convierte una cadena JSON a un objeto JavaScript.

Sintaxis

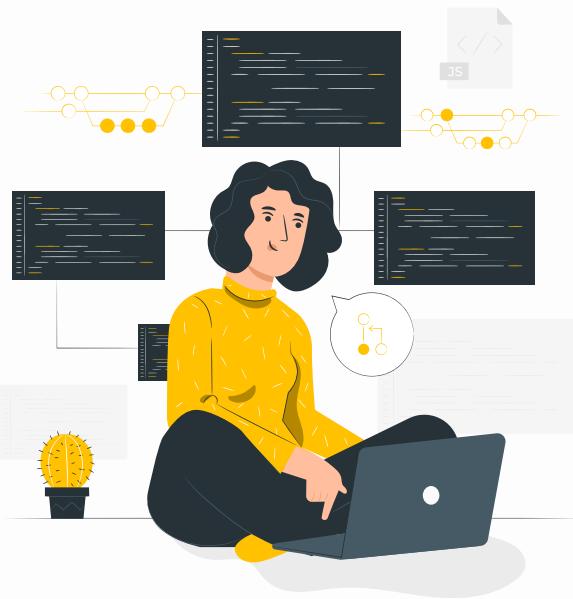
```
JSON.parse(stringJSON, reviver)
```

- **stringJSON**: Una cadena de texto en formato JSON válida.

- **reviver**(*opcional*): Una función que puede modificar los valores antes de devolver el objeto

```
const jsonToConvert = {  
    "name": "Juan",  
    "age": 30,  
    "isStudent": true,  
    "courses": ["Matemáticas", "Programación"]  
}  
const data = JSON.parse(jsonToConvert);  
  
console.log(data.name); // Juan
```

Uso del parámetro reviver



El parámetro reviver permite aplicar una transformación a los valores del objeto antes de que sea devuelto.

```
const data = JSON.parse(jsonToConvert, () => {  
  return key === 'isStudent' ? true : key;  
});
```

En el ejemplo anterior utilizamos una función de callback donde durante el proceso de conversión a objeto literal, cambia el valor de la key `isStudent` a `true`.

JSON.stringify()

Convierte un objeto JavaScript a una cadena JSON.

Sintaxis

```
JSON.stringify(valor, replacer,  
espacio)
```

- **valor**: El objeto que se desea convertir en JSON.
- **replacer (opcional)**: Una función o un array que determina qué propiedades incluir en la conversión.
- **espacio (opcional)**: Un número o cadena que define la indentación en la salida para hacerla más legible.

Uso del parámetro espacio

Este parámetro permite generar una salida más legible agregando espaciados:

```
const obj = { name: "Luis", age: 40,  
city: "Madrid" };  
const jsonString = JSON.stringify(obj,  
null, 2);  
// Indentación de 2 espacios  
console.log(jsonString);  
/*  
{  
  "name": "Luis",  
  "age": 40,  
  "city": "Madrid"  
}  
*/
```

- ❑ Estos métodos son fundamentales para el manejo de datos en aplicaciones web, facilitando el intercambio de información entre el servidor y el cliente.

Filesystem

Filesystem en Node.js

El módulo **Filesystem (fs)** de Node.js permite interactuar con el sistema de archivos del servidor, proporcionando funciones para leer, escribir, modificar y eliminar archivos y directorios. Es una herramienta fundamental en el desarrollo backend, especialmente cuando se necesita manipular datos de manera persistente.

Importación

```
import fs from 'fs';
```

Formas de trabajo

Síncrono (bloquea ejecución) y Asíncrono (en segundo plano).

Operaciones

Leer, escribir, modificar y eliminar archivos y directorios.

Lectura de Archivos

Síncrono (`fs.readFileSync`)

```
import fs from 'fs';

const data =
  fs.readFileSync('data.txt', 'utf8');
  console.log(data);
```

En este caso, la ejecución del código se detiene hasta que el archivo sea leído por completo.

- `readFileSync` recibe 2 parámetros, el primero es la ruta a la ubicación de nuestro archivo dentro del servidor y el segundo es la codificación de caracteres, en este caso 'utf8' con aceptación de caracteres Unicode.

Lectura de Archivos

Asíncrono (`fs.readFile`)

```
fs.readFile('data.txt', 'utf8',
(err, data) => {
  if (err) {
    console.error('Error al leer
el archivo:', err);
    return;
  }
  console.log(data);
});
```

El archivo se lee en segundo plano y el resultado se devuelve a través de un callback.

- ❑ `readFile` a diferencia del anterior, recibe 3 parámetros, los 2 primeros son iguales a `readFileSync` mientras que el tercero es una función del callback que se ejecutará una vez leído el archivo devolviendo su contenido o un error en caso de existir.

Otros Métodos de Filesystem

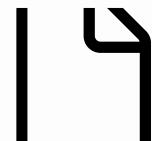
Además de métodos de lectura, **filesystem** posee otros que nos permiten escribir (modificar), eliminar y crear archivos e incluso directorios (carpetas) completos.



Escribir

fs.writeFileSync y **fs.writeFile**

- Permite sobreescribir el contenido completo de un archivo.



Agregar

fs.appendFileSync y **fs.appendFile**

- Agregan contenido a la última fila de un archivo.



Eliminar

fs.unlinkSync y **fs.unlink**

- Elimina archivos

Lectura asíncrona o síncrona

- Se recomienda **usar las versiones asíncronas** (fs.readFile, fs.writeFile, etc.) en aplicaciones web o servidores, ya que evitan bloqueos en la ejecución y mejoran el rendimiento.
- Las funciones **síncronas** (fs.readFileSync, fs.writeFileSync, etc.) son útiles en scripts pequeños o cuando es necesario garantizar que una tarea se complete antes de continuar.

*Nota: en ocasiones el uso de **fs** se complementa con el módulo **path** y la variable global **__dirname** para poder obtener las rutas a los archivos de forma dinámica.

Recordemos que **__dirname** no está disponible directamente cuando utilizamos ESModules, por lo que con este pequeño snippet de código podemos resolverlo:

```
const __dirname = import.meta.dirname;
```

Modelos

Modelos en API Rest



Representaciones de entidades de datos en la aplicación.



Función

Actúan como capa intermedia entre lógica de negocio y fuente de datos.



Objetivo

Encapsular lógica relacionada con datos para modularidad y mantenibilidad.

Por ejemplo, si estamos construyendo una aplicación para gestionar usuarios, un modelo de Usuario definiría cómo se estructura la información de un usuario (nombre, correo, contraseña, etc.) y cómo se interactúa con esa información (crear, leer, actualizar o eliminar).

Creación de un Modelo en Node.js

Definir estructura

Crear clase o objeto con propiedades y métodos.

Implementar métodos

Desarrollar funciones para manipular datos.

Conectar con datos

Utilizar Filesystem para leer/escribir en JSON.



Servicios vs Modelos

En una arquitectura bien estructurada, los modelos y los servicios tienen responsabilidades claramente definidas, pero su interacción puede generar cierta confusión debido a su similitud en la declaración de métodos.

Modelos

- Los modelos se encargan de interactuar directamente con los datos. En este caso, el modelo Product maneja la lectura y escritura del archivo JSON, simulando el acceso a una base de datos.
- Su objetivo es abstraer la lógica de acceso a los datos, proporcionando métodos como getAllProducts, getProductById, saveProduct y deleteProduct.
- Los modelos no deben contener lógica de negocio; su función es simplemente gestionar los datos.

Servicios

- Los servicios actúan como una capa intermedia entre los controladores y los modelos. Su objetivo es manejar la lógica de negocio y orquestar las operaciones necesarias para cumplir con los requisitos de la aplicación.
- En el ejemplo, el servicio products.service.js utiliza los métodos del modelo Product para obtener, crear o eliminar productos.
- Los servicios pueden enriquecer o transformar los datos antes de pasarlos al controlador. Por ejemplo, podrían combinar datos de múltiples modelos, aplicar reglas de negocio o validaciones adicionales.

Casos de Uso de Servicios



Lógica de Negocio

Aplicar reglas complejas que no pertenecen al modelo.



Combinación de Datos

Unir información de múltiples modelos.



Validaciones

Realizar comprobaciones adicionales antes de operaciones.



Reflexión y Próximos Pasos

Importancia de modelos

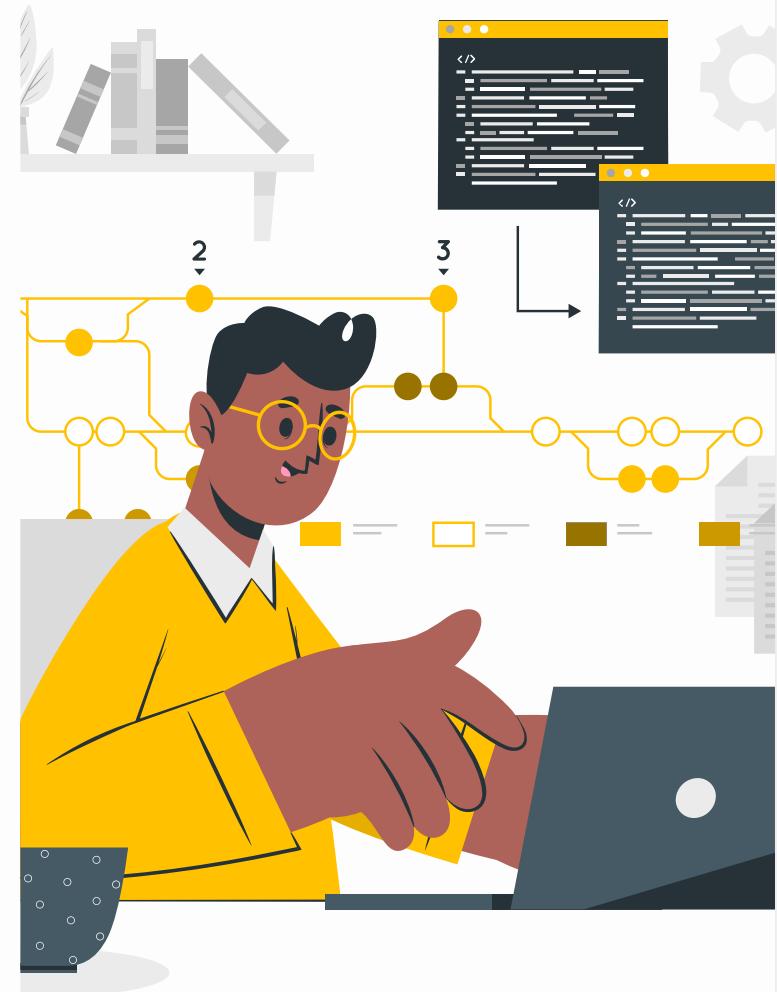
¿Cómo contribuyen a la organización del código?

Ventajas de JSON

¿Por qué usar archivos JSON como base de datos?

Escalabilidad

¿Cómo migrar a una base de datos relacional o NoSQL?



Materiales y recursos adicionales

- 1 Documentación Oficial de JSON:

<https://www.json.org/json-es.html>

- 2 Documentación de Node.js sobre el módulo FS:

<https://nodejs.org/api/fs.html>





Ejercicio Práctico

Ejercicio Práctico

Obligatorio

Organización de la Capa de Modelos en tu API

Sabrina y Matías regresan para revisar tu progreso. Esta vez, Sabrina es quien toma la palabra con entusiasmo:

"¡Tu organización ha mejorado muchísimo! Ya tienes rutas bien definidas, controladores claros y una capa de servicios eficiente. Ahora es momento de llevar la estructura un paso más allá."



Matías asiente y añade con una mirada seria:

"Hasta ahora, has estado manejando datos simulados directamente en la capa de servicios, pero en un proyecto real, los datos suelen almacenarse en bases de datos o archivos estructurados. Por eso, vamos a introducir la capa de modelos, que centralizará la gestión de datos."



Ejercicio Práctico

Obligatorio

Misión: Matías concluye con una mirada confiada: "Si logras completar este desafío, habrás dado un paso gigante hacia el desarrollo backend profesional. ¡Es hora de escribir código limpio y estructurado!"

1. Crear la capa de modelos

- Organiza los datos de tu aplicación en archivos JSON dentro de una carpeta llamada data.
- Migra los datos simulados que estaban en los servicios hacia estos archivos.
- Asegúrate de que la estructura de los archivos JSON sea clara y representativa de la información que manejas.
- Crea archivos para los modelos de tu aplicación y crea los métodos necesarios para interactuar con los JSON de datos.

2. Modificar los servicios para interactuar con los modelos

- Ajusta la capa de servicios para que en lugar de devolver datos directamente desde el código lo haga utilizando los métodos creados en los modelos.
- Asegúrate que los controladores sigan obteniendo los datos correctamente desde los servicios.

