Persistencia

Persistencia en archivos planos

{desafío} latam_

Implementar la persistencia de objetos en una aplicación web utilizando archivos de texto plano para resolver un problema acorde al entorno Node.js.

- Unidad 1:
  Introducción a Node
- Unidad 2:
  Node y el gestor de paquetes
- Unidad 3:
  Persistencia

Te encuentras aquí

{desafío}
latam_

¿Qué aprenderás en esta sesión?

- *Reconocer los módulos requeridos para trabajar con archivos en un proyecto acorde al entorno Node.js.*
- *Utilizar instrucciones de lectura y escritura de objetos JSON en archivos de texto para dar solución a un problema acorde al entorno Node.js.*

{desafío} latam_

¿Alguna vez has oído hablar de Node.js y cómo se manejan los archivos en él para almacenar y leer información?

/* Concepto de persistencia */

Concepto de persistencia

- Podemos entender a la persistencia como la capacidad que tiene un sistema, aplicación o software para almacenar los datos que este gestiona de manera íntegra, asegurando que permanezcan a lo largo del tiempo y el espacio.

- Dependiendo de qué sistema se esté hablando, la forma de persistir los datos o crear archivos va a variar. Por ejemplo, en el caso de las fotografías digitales, estas suelen almacenarse en archivos únicos que contienen todos los datos (*.png, *.jpeg, *.bmp, etc.), en cambio, en sistemas complejos los datos pueden estar almacenados en archivos específicos y de manera separada, para satisfacer diferentes fines, como la seguridad y una mayor integridad de los datos.

Persistencia

Persistencia de datos en programación

¿Cuáles son las herramientas que debo usar en programación para persistir información? En Node, disponemos de un módulo integrado que permite trabajar con el sistema de archivos llamado fs. Este módulo, al ser integrado dentro del motor de Node, no es necesario que sea instalado.

En nuestras aplicaciones, podemos integrar la persistencia almacenando los datos que controlamos en nuestro código o el que ingresa un usuario desde el cliente como archivos locales en el servidor. El formato por convención de almacenamiento será JSON, por lo que deberemos ocupar los métodos que dispone Javascript correspondientes a este formato.

Demostración
"Mi primer JSON"

Ejercicio guiado

Mi primer JSON

En Javascript, contamos con el método JSON.stringify() para convertir un dato de tipo objeto a un string con formato JSON, y esto es justo lo que necesitamos para hacer nuestro primer JSON, el cual será almacenado en un archivo dentro del servidor usando el método writeFileSync de File System.

Dado esto, se solicita almacenar un objeto en un archivo con formato JSON, con datos referentes a un auto que tengamos en nuestro código.

Ejercicio guiado

Mi primer JSON

Para lo anterior, sigue los siguientes pasos:

- **Paso 1:** Crear un archivo index.js para escribir el programa.
- **Paso 2:** Importar el módulo fs.
- **Paso 3:** Crear un objeto con propiedades que almacenen nombre de una marca y un modelo de auto.
- **Paso 4:** Usar el método writeFileSync para guardar en un archivo llamado “MiAuto.json” el objeto creado en formato JSON.

```
// Paso 2
const fs = require('fs')
// Paso 3
let miAuto = {
    marca : 'Lamborghini',
    modelo: ' AVENTADOR S',
}
// Paso 4
fs.writeFileSync('MiAuto.json', JSON.stringify(miAuto))
```

Ahora ejecuta este programa con la terminal usando el comando “node index.js” y deberás ver un archivo nuevo en la carpeta, en la que te encuentres con el nombre indicado y como contenido nuestro objeto en formato JSON.

Persistencia

Leyendo mi primer JSON

- Una de las operaciones más recurrentes que se realizan al utilizar archivos en cualquier aplicación es la de lectura.
- En Node, disponemos del método fs.ReadFile() que permite realizar lecturas asíncronas, no bloqueantes. Por otro lado, tenemos al método fs.ReadFileSync() para realizar operaciones que bloquean el flujo de ejecución.
- Utilizaremos este método por motivos académicos para lograr un código más vertical y legible. Sin embargo, en el mundo laboral, Node se ha popularizado por su naturaleza asíncrona e instrucciones “non-blocking”, por lo tanto utilizamos idealmente sus herramientas asíncronas para no detener el flujo de ejecución.

Demostración
"Leyendo mi primer JSON"

Ejercicio guiado

Leyendo mi primer JSON

Ahora que tenemos un archivo propio en formato JSON, probemos si podemos leer su contenido y más importante todavía, poder manejarlo como objeto.

Para esto, elimina el código escrito anteriormente en el archivo index.js, dejando solo la importación de FileSystem y prosigue con los siguientes pasos en donde el objetivo será imprimir por consola el modelo de nuestro auto:

- Paso 1: Usar el método ReadFile() para leer el archivo MiAuto.json.
- Paso 2: Usar el método JSON.parse para parsear la data obtenida del archivo MiAuto.json.

- **Paso 3:** Imprimir por consola el modelo del auto

```
const fs = require("fs");
// Paso 1
fs.readFile("MiAuto.json", "utf8", function (e,
data) {
  // Paso 2
  let miAuto = JSON.parse(data);
  // Paso 2
  console.log(miAuto.modelo);
});
```

```
$ node index.js
AVENTADOR S
```

/* Servidores y persistencia */

Servidores y persistencia

- En la industria del software, los datos no se escriben directamente en las aplicaciones, sino que provienen de usuarios interactuando con aplicaciones cliente.

- Los usuarios depositan información, desencadenando funciones que consultan un backend. Combinar persistencia de datos en archivos JSON y rutas de servidor facilita que los usuarios manipulen datos almacenados en el servidor.

- En el ejercicio siguiente, usaremos métodos síncronos de File System, aunque en la práctica se prefieren métodos asíncronos para permitir consultas simultáneas de múltiples clientes en el servidor.

Demostración
“Registro de usuarios”

Ejercicio guiado

Registro de usuarios

- Crear un servidor que disponibilice una ruta para la recepción de query Strings con información de un usuario a registrar, por ejemplo:

http://localhost:3000/nuevoUsuario?name=Akira&lastname=Toriyama&email=akayama@example.com&password=1234

- Esta información deberá ser almacenada en un documento con formato JSON.

Ejercicio guiado

Registro de usuarios

Dado lo anterior, sigue los siguientes pasos para llegar a la solución:

- **Paso 1:** Importa File System, express y levanta un servidor en el puerto 3000
- **Paso 2:** Crea una ruta GET /nuevoUsuario
- **Paso 3:** Importar el módulo fs
- **Paso 4:** Crear un servidor en el puerto 3000

```
// Paso 1
const fs = require("fs")
const express = require('express');
const app = express();
app.listen(3000)

// Paso 2
app.get("/nuevoUsuario", (req, res) => {
    // Paso 3
    const { name, lastname, email, password } =
req.query
    const usuario = {
        name,
        lastname,
        email,
        password,
    };
    // Paso 4
    fs.writeFileSync("usuario.json",
JSON.stringify(usuario));
    res.send("Usuario almacenado con éxito");
})
```

/* Almacenando varios usuarios */

Almacenando varios usuarios

La forma de acumular instancias en nuestras aplicaciones, comúnmente ha sido por medio de arreglos de objetos y esto seguirá siendo así al momento de persistir la información. Es un poco abstracto de verlo al comienzo, pero en palabras sencillas necesitamos seguir la siguiente receta:

1. Tener de antemano un JSON que contenga un arreglo de usuarios.

1. Cada vez que necesitemos agregar un usuario nuevo deberemos consumir la data almacenada en el archivo JSON y formatearla con el JSON.parse() para poder usarla como un objeto.

1. Agregar al arreglo el usuario recibido en la consulta.

1. Sobrescribir el documento JSON con la data del objeto luego de haber agregado el usuario nuevo.

Almacenando varios usuarios

Entendiendo lo anterior, procedamos con la modificación de la ruta que creamos tomando en cuenta la receta, pero antes crea manualmente un archivo JSON llamado Usuarios.json con el siguiente código:

```
{
  "usuarios": []
}
```

Almacenando varios usuarios

Ahora sigue los siguientes pasos para lograr el almacenamiento acumulativo de usuarios:

- **Paso 1:** Almacenar en una variable la data del archivo “Usuarios.json”
- **Paso 2:** Almacenar en una variable la propiedad “usuarios” de la variable “data“
- **Paso 3:** Agregar al arreglo “usuarios” el usuario recibido en query Strings
- **Paso 4:** Sobrescribir el archivo “Usuarios.json” con la data modificada

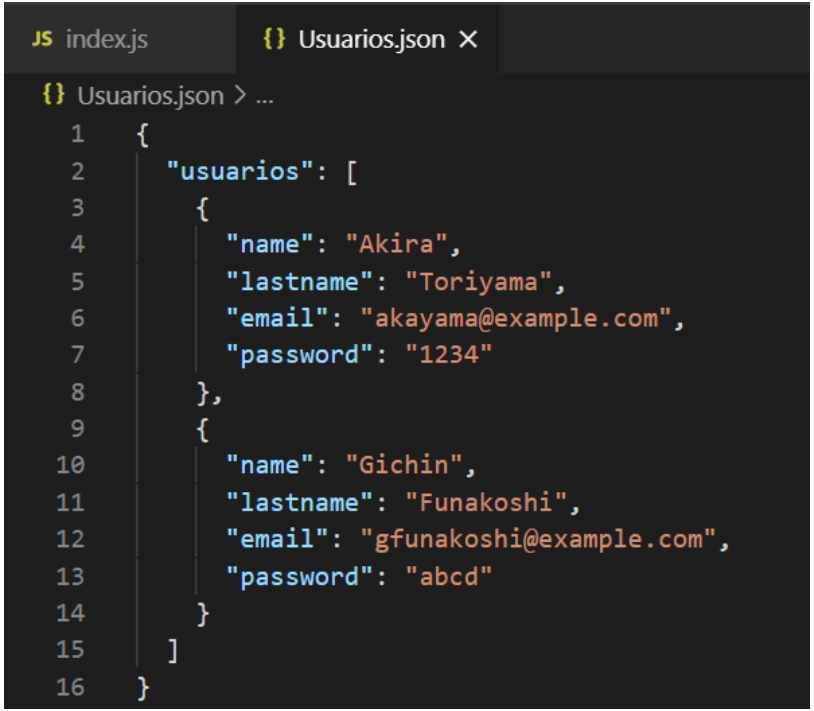
```
app.get("/nuevoUsuario", (req, res) => {
    const { name, lastname, email, password } = req.query
    const usuario = {
        name,
        lastname,
        email,
        password,
    };
    // Paso 1
    const data = JSON.parse(fs.readFileSync("Usuarios.json", "utf8"));
    // Paso 2
    const usuarios = data.usuarios;
    // Paso 3
    usuarios.push(usuario);
    // Paso 4
    fs.writeFileSync("Usuarios.json", JSON.stringify(data));
    res.send("Usuario almacenado con éxito");
})
```

Almacenando varios usuarios

Ahora solo queda hacer la prueba, para esto consulta el servidor con las siguientes URLs, las cuales contienen información de 2 usuarios diferentes.

http://localhost:3000/nuevoUsuario?name=Gichin&lastname=Funakoshi&email=gfunakoshi@example.com&password=abcd

http://localhost:3000/nuevoUsuario?name=Akira&lastname=Toriyama&email=akayama@example.com&password=1234



Título

Subtítulo

- La persistencia se refiere a la capacidad de sistemas, aplicaciones o software para almacenar datos de forma duradera.

- En programación, Node.js utiliza el módulo 'fs' para trabajar con el sistema de archivos. Los datos pueden almacenarse como archivos locales en el servidor en formato JSON. Para esto, se utiliza JSON.stringify() para crear un JSON y fs.writeFileSync() para guardar el objeto en un archivo.

- La lectura de archivos también es vital. Se puede usar fs.readFile() de forma asíncrona o fs.readFileSync() de manera bloqueante. Los datos leídos, generalmente en formato JSON, se pueden parsear usando JSON.parse() para manipularlos como objetos y realizar acciones como imprimir propiedades específicas.

¿Cómo se aplica la persistencia de información en un servidor y qué pasos debes seguir para lograrlo?

Próxima sesión...

- *Guía de ejercicios - Persistencia*

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales