Prueba Técnica Full Stack para Proper Business Solutions

Autor: Mario Castro (mariocastro.pva@gmail.com)

1. Agregar un registro nuevo en la tabla:

- a. Para agregar un nuevo registro que pertenezca a la familia de Gato, simplemente necesitamos asignarle un `idPadre` que coincida con el `id` del Gato (que es 2).
- b. Similarmente, para agregar un nuevo registro que pertenezca a la familia de Fido, necesitamos asignarle un `idPadre` que coincida con el `id` de la familia de Fido (que es un perro, por lo que su familia es la de `id` 3).

implementación:

```
// Suponiendo que la tabla es un array de objetos
let tabla = [
    {"Id": 1, "Nombre": "Mascotas", "idPadre": 0},
    {"Id": 2, "Nombre": "Gato", "idPadre": 1},
    {"Id": 3, "Nombre": "Perro", "idPadre": 1},
    {"Id": 4, "Nombre": "Plantas", "idPadre": 0},
    {"Id": 5, "Nombre": "Árbol", "idPadre": 4},
    {"Id": 6, "Nombre": "Flores", "idPadre": 3},
    {"Id": 7, "Nombre": "Micu", "idPadre": 2},
    {"Id": 8, "Nombre": "Sasy", "idPadre": 2},
    {"Id": 9, "Nombre": "Fido", "idPadre": 3},
    {"Id": 10, "Nombre": "Bobby", "idPadre": 3},
    {"Id": 11, "Nombre": "Roble", "idPadre": 5}
];

// Agregar un registro para la familia de Gato
let nuevoGato = {"Id": 12, "Nombre": "NuevoGato", "idPadre": 2};
tabla.push(nuevoGato);

// Agregar un registro para la familia de Fido
let nuevoFido = {"Id": 13, "Nombre": "NuevoFido", "idPadre": 3};
```

```
tabla.push(nuevoFido);

console.log(tabla);
```

2. Existe un registro que no coincide con la interpretación esperada:

El registro que no coincide con la interpretación esperada es el registro de "Flores". Según la interpretación esperada, "Flores" debería estar bajo la categoría de "Plantas", sin embargo, en la tabla original, tiene un `idPadre` que coincide con el `id` de "Perro" (que es 3). Esto parece ser un error en los datos.

A partir de este momento se tomara el conjunto de datos, corrigiendo el `idPadre` de “Flores” a 4

3. Función recursiva para entregar un listado en consola:

```
// Definición de la función recursiva
function listarFamilias(familias, parentId = 0, level = 0) {
    const indent = "  ".repeat(level);
    const hijos = familias.filter(familia => familia.idPadre === parentId);

    hijos.forEach(hijo => {
        console.log(`${indent}-${hijo.Nombre}`);
        listarFamilias(familias, hijo.Id, level + 1);
    });
}

// Datos de ejemplo (la tabla de la base de datos)
// incluyendo la correccion del punto 2
const familias = [
    { Id: 1, Nombre: "Mascotas", idPadre: 0 },
    { Id: 2, Nombre: "Gato", idPadre: 1 },
    { Id: 3, Nombre: "Perro", idPadre: 1 },
    { Id: 4, Nombre: "Plantas", idPadre: 0 },
    { Id: 5, Nombre: "Árbol", idPadre: 4 },
    { Id: 6, Nombre: "Flores", idPadre: 4 },
    { Id: 7, Nombre: "Micu", idPadre: 2 },
    { Id: 8, Nombre: "Sasy", idPadre: 2 },
    { Id: 9, Nombre: "Fido", idPadre: 3 },
    { Id: 10, Nombre: "Bobby", idPadre: 3 },
    { Id: 11, Nombre: "Roble", idPadre: 5 }
];

// Llamada a la función para listar las familias
listarFamilias(familias);
```

Explicación del código:

Este código en JavaScript es una implementación de una función recursiva llamada `listarFamilias` que se encarga de imprimir en la consola una lista jerárquica de familias. Cada familia puede tener hijos que también son familias, lo que permite representar una estructura de árbol de datos.

Ahora, vamos a analizar el código paso a paso:

1. Se define la función `listarFamilias` que toma tres parámetros:
    - `familias`: un array que contiene objetos representando las familias y sus relaciones.
    - `parentId`: el identificador del padre de la familia actual. Por defecto, su valor es 0, lo que significa que estamos buscando las familias que no tienen un padre (las raíces del árbol).
    - `level`: el nivel de profundidad en la jerarquía. Por defecto, su valor es 0.
2. Se calcula el valor de `indent` que es una cadena de espacios que se utiliza para indentar la salida en la consola. La cantidad de espacios es igual a `level` multiplicado por 2 (2 espacios por nivel).
3. Se filtran las familias que tienen un `idPadre` igual al `parentId` pasado como argumento. Esto devuelve un array con las familias que son hijos de la familia actual.
4. Se itera sobre cada hijo encontrado y se realiza lo siguiente:
    - Se imprime en la consola el nombre del hijo, precedido por un guión y con la indentación adecuada.
    - Se llama recursivamente a la función `listarFamilias` pasando como argumentos el array `familias`, el `id` del hijo actual como nuevo `parentId` y aumentando `level` en 1. Esto permite explorar la estructura de árbol de manera recursiva.
5. Se definen los datos de ejemplo `familias`, que representan una estructura jerárquica de familias, donde algunas tienen padres (idPadre diferente de 0) y otras no.
6. Se realiza una llamada inicial a la función `listarFamilias` pasando el array de `familias`. Esto inicia el proceso de impresión de la jerarquía de familias en la consola, comenzando desde las raíces.

En resumen, este código implementa una función recursiva para imprimir en la consola una lista jerárquica de familias, siguiendo la estructura de un árbol donde cada familia puede tener hijos que también son familias. La recursión permite explorar profundamente esta estructura, imprimiendo cada familia en su nivel correspondiente de indentación.

4. Idea para trabajar de forma diferente la estructura o lógica de la tabla:

Una forma alternativa de estructurar la tabla podría ser utilizando una estructura de árbol, donde cada nodo tenga una lista de hijos. Esto facilitaría la manipulación y la navegación dentro del árbol. Sin embargo, la implementación y el acceso a los datos podrían ser un poco más complejos.

```
// Definir una clase para representar cada nodo de la jerarquía
class Nodo {
    constructor(id, nombre) {
        this.id = id;
```

```
        this.nombre = nombre;
        this.hijos = [];
    }
}

// Función para agregar un nuevo nodo a la jerarquía
function agregarNodo(padre, id, nombre) {
    const nuevoNodo = new Nodo(id, nombre);
    padre.hijos.push(nuevoNodo);
    return nuevoNodo;
}

// Función para imprimir la jerarquía como se espera
function imprimirJerarquia(nodo, nivel = 0) {
    console.log("  ".repeat(nivel) + "-" + nodo.nombre);
    nodo.hijos.forEach(hijo => imprimirJerarquia(hijo, nivel + 1));
}

// Crear nodos iniciales para la jerarquía
const mascotas = new Nodo(1, "Mascotas");
const gato = agregarNodo(mascotas, 2, "Gato");
const perro = agregarNodo(mascotas, 3, "Perro");
const plantas = new Nodo(4, "Plantas");
const arbol = agregarNodo(plantas, 5, "Árbol");
const flores = agregarNodo(plantas, 6, "Flores");
const micu = agregarNodo(gato, 7, "Micu");
const sasy = agregarNodo(gato, 8, "Sasy");
const fido = agregarNodo(perro, 9, "Fido");
const bobby = agregarNodo(perro, 10, "Bobby");
const roble = agregarNodo(arbol, 11, "Roble");

// Agregar nuevos nodos
const nuevoGato = agregarNodo(gato, 12, "Nuevo Gato");
const nuevoFido = agregarNodo(perro, 13, "Nuevo Fido");

// Imprimir la jerarquía como se espera
console.log("Interpretación esperada:");
imprimirJerarquia(mascotas);
imprimirJerarquia(plantas);
```