Servidor Básico Unidad 1

Nombre: Amanda Cristina Flores Feijoo
de expediente: 223D1626

En mi código crea un servidor HTTP en Node.js con datos de un archivo JSON llamado pokedex.json.

Las principales características:

1) Se utilizan los módulos de http, url y fs de Node.js. para crear el servidor, analizar las URL de las solicitudes y leer archivos del sistema de archivos.

2) Se lee el archivo pokedex.json de forma sincrónica que lo convierte en un objeto JavaScript utilizando JSON.parse.

3) Se crea un servidor HTTP que escucha las solicitudes entrantes. Para cada solicitud, extrae el pathName de la URL y lo decodifica.

```
const server = http.createServer((req, res) => {
    let pathName = url.parse(req.url,true).pathname.slice(1);
    pathName = decodeURIComponent(pathName);
```

El decodeURIComponent esto me ayudo para poder asegurar que cualquier carácter especial en la URL (como los caracteres utilizados en los nombres de Pokémon en chino o japonés) se interprete correctamente.

En conclusión, independientemente del idioma del nombre del Pokémon (inglés, chino, japonés, ect.) decodeURIComponent, permite que la búsqueda funcionara correctamente.

4) Intenta encontrar un Pokémon en la pokedex por su ID. Sino es un número, intenta encontrar un Pokémon por su nombre. Si pathName es un número, el servidor busca un Pokémon con ese ID. Si pathName no es un número, el servidor busca un Pokémon con ese nombre. La búsqueda de nombres no distingue entre mayúsculas y minúsculas.

```
// Busca por ID
if (!isNaN(pathName)) {
    pokemon = pokedex.find(p => p.id === Number(pathName));
}
// Busca por nombre
else {
    pokemon = pokedex.find(p =>
        Object.values(p.name).some(n => n.toLowerCase() === pathName.toLowerCase())
    );
}
```

En estas líneas de códigos verifica isNaN es una función de JavaScript que devuelve true si el valor que se pasa no es número. El operador! niega el resultado, por lo que, !isNaN(pathName) será true si pathName es un número.

En el siguiente bloque de código es la condición que debe cumplir un Pokémon por el método find, que devuelve un array con todos los valores name.

5) Luego si ha encontrado un Pokémon, el servidor responde con un objeto JSON que contiene el tipo de Pokémon y sus estadísticas base. Si no se encuentra el Pokémon, el servidor responde con el mensaje “Pokemon not found”.

   HP: Los puntos de salud del Pokémon.
   Attack: El valor de ataque del Pokémon.
   Defense: El valor de defensa del Pokémon.
   Sp.Attack: Valor de ataque especial del Pokémon.
   Sp.Defense: El valor de defensa especial del Pokémon.
   Speed: La velocidad del Pokémon.

Aquí tuve que poner el código de esta manera:

```
  "Sp. Attack": pokemon.base["Sp. Attack"],
  "Sp. Defense": pokemon.base["Sp. Defense"],
```

Se me presento dificultad esta parte, porque no me aparecían estos detalles, y tuve cambiar el código, porque las claves Sp. Attack y Sp. Defense contienen puntos y espacio, y esto se consideran que son caracteres especiales. Razón por la que van entre comillas.

6) Luego se crea este código:

```
  res.writeHead(200, {'Content-Type': 'application/json'});
  res.end(JSON.stringify(response));
else {
  res.writeHead(404, {'Content-Type': 'text/plain'});
  res.end('Pokemon not found');
```

Con el objetivo que, si encuentra un Pokémon, crea un objeto de respuesta con los datos del Pokémon con un código de estado HTTP 200. si no encuentra el Pokémon, envía una respuesta con un código de estado HTTP 404 y un mensaje de error (“Pokémon not found”).

7) Por último, el servidor comienza a escuchar en el puerto 3000 y muestra un mensaje en la consola está listo para recibir solicitudes.