Servidor Básico

Explicando el código

```
const http = require('http')
const https = require('https')
```

- Nos traemos el http para poder crear el servidor y el https para poder llamar al json y sacar los datos de él.

```
const fetchPokemonData = async () => {
    return new Promise((resolve, reject) => {
        https.get('https://raw.githubusercontent.com/fanzeyi/pokemon.json/master/pokedex.json', (res) => {
            let data = ''

            res.on('data', (chunk) => {
                // console.log(data)
                data += chunk
            })

            res.on('end', () => {
                resolve(JSON.parse(data))
            })

        }).on('error', (err) => {
            reject(err)
        })
    })
}
```

- Creamos la función fetchPokemonData, la cual nos saca los datos del json.
- Retorna una promesa en la que llamamos al json, e inicializamos una variable data vacía, que cuando la respuesta sea data, se van añadiendo los chunks de datos a esta variable para poder almacenar todos los datos de los Pokémon.
- Si la respuesta termina (end), retorna los datos almacenados en un objeto js para poder utilizarlo en nuestro código.
- Pero si esta petición no funciona, devuelve un error, por lo que se usa reject.

```
const handleRequest = async (req, res) => {
    const pokemonName = decodeURI(req.url.substring(1)).toLowerCase()
    const pokedexData = await fetchPokemonData()

    const pokemonData = pokedexData.find(pokemon => pokemon.name.english.toLowerCase() === pokemonName ||
        pokemon.name.japanese === pokemonName ||
        pokemon.name.chinese === pokemonName ||
        pokemon.name.french.toLowerCase() === pokemonName ||
        pokemon.id.toString() === pokemonName)

    if (pokemonData) {
        const response = {
            'Tipo': pokemonData.type,
            'HP': pokemonData.base.HP,
            'Attack': pokemonData.base.Attack,
            'Defense': pokemonData.base.Defense,
            'Sp. Attack': pokemonData.base['Sp. Attack'],
            'Sp. Defense': pokemonData.base['Sp. Defense'],
            'Speed': pokemonData.base.Speed,
            'Base Total': pokemonData.base.HP + pokemonData.base.Attack +
            pokemonData.base.Defense + pokemonData.base['Sp. Attack'] +
            pokemonData.base['Sp. Defense'] + pokemonData.base.Speed
        }
        res.writeHead(200, { 'Content-Type': 'application/json' })
        res.end(JSON.stringify(response, null, 2))
        } else {
            res.writeHead(404, { 'Content-Type': 'text/plain' })
        res.end('Pokémon no encontrado')
        }
}
```

- Después, queremos realizar una función que nos permita manejar los datos obtenidos, por lo que creamos handleRequest.
- Lo primero que hacemos es crear una constante que obtenga el substring introducido en la url localhost:3000 (hemos inicializado el servidor en el puerto 3000 como veremos más adelante), como por ejemplo localhost:3000/Pikachu. Este 'Pikachu' es el dato almacenado en PokemonName. Luego creamos la constante pokedexData que es la nos trae los datos recogidos con fetchPokemonData.
- De nuevo creamos una constante más llamada pokemonData para que nos diga si el valor de pokemonName, está en el valor de algún Pokémon como nombre en inglés, francés, japonés o chino. También le pedimos que, si se trata de un número, nos localice el Pokémon por id.
- Para que se pueda buscar por id, hemos tenido que decirle que al buscar en el pokemon.id se pase a string con toString() ya que el valor del id es un número, pero el valor de pokemonName introducido en la url es un string.
- También hemos ido un paso más allá y hemos pedido que el valor introducido en la url se pase a minúscula con la función toLowerCase() para que al escribir el nombre del Pokémon del que queramos sus datos, no importe si está en minúscula o mayúscula, ya que en el json del que sacamos la información, los nombres de los Pokémon empiezan todos por mayúscula.
- Así mismo, para que esto funcione, también tenemos que decirle que cuando compruebe los nombre es inglés y francés, se pasen a minúscula para comprobar que sean igual a pokemonName.
- Si el nombre o id introducidos en la url coinciden con el de alguno de los Pokémon, entonces hemos escrito una respuesta para que salga en la web los datos del Pokémon indicado (he

querido añadir los STATS base totales para añadir algo más), pero si no coinciden, hemos indicado que nos saque un error por pantalla.

```
const server = http.createServer(handleRequest);

server.listen(3000, () => {
    console.log('Servidor escuchando en el puerto 3000');
})
```

- Y, por último, creamos la constante server en la que creamos con http el servidor con la función handleRequest.
- También como dijimos, inicializamos el server en el puerto 3000.

Conclusiones y observaciones

He podido observar que los datos en pantalla salen como si fueran sacados de un JSON y quizás me hubiera gustado sacarlos por pantalla simplemente que me pusiera:

Tipo: Tipo/Tipo2

Ataque: …

Defensa: …

Sin llaves ni signos de puntuación, es algo que me gustaría aprender a hacer, quizás solo tengo que utilizar mis conocimientos de JS para poder hacerlo así.

Otra cosa que he visto es que, al sacar los datos de las estadísticas de los Pokémon, cuando sacaba el de Ataque y Defensa Especiales, al contener dos palabras no bastaba con hacerlo como el resto, como, por ejemplo, HP que ponía pokemonData.base.HP y tenía que ponerlo así pokemonData.base['Sp. Attack']. Es un dato curioso que acabo de ver por primera vez.

La verdad que este ejercicio me ha ayudado bastante a comprender mejor los conceptos del Backend y organizarlos en mi cabeza, y creo que he sido capaz de defenderlo bien e incluso ir un paso más allá en algunos aspectos del ejercicio.