HINTS

MANEJANDO ERRORES EN ASYNC/AWAIT

Al usar await, cuando una Promesa se resuelve de manera exitosa, entonces ésta devuelve el resultado. Pero, en el caso de que no se resuelva la Promesa, el código arrojaría una excepción, tal como si hubiera una declaración throw. Si deseamos manejar estos errores, y personalizar nuestros mensajes, podemos valernos del `try catch`, los objetos `Error`, y la palabra clave `throws`. El siguiente ejemplo muestra como podemos usar estos 3 elementos. Te recomendamos usar el código para observar cómo se comporta este programa en tu computadora, y luego entraremos más en detalle sobre cómo funciona.

```
var b = false;
const miPromesa = new Promise((resolve, reject) => {
    if (b) {
        resolve('listo!');
    } else {
        reject('error!')
    }
});

async function haceAlgo() {
    try {
        let respuesta = await miPromesa();
        console.log(respuesta)
    } catch (err) {
        throw new Error('Error manejado en el try catch :)');
    }
}
haceAlgo();
```

En este caso, la Promesa `miPromesa` está destinada a fallar, pues `b` es `false`. Esto hace que no se resuelva, sino que devuelve el mensaje de rechazo. Ahora, considerando la función async `hacerAlgo()`, aquí definimos nuestra estructura de control de flujo: el `try catch`. Dentro del `catch` podemos manejar el error, y para desplegar por pantalla un nuevo mensaje, utilizamos `throw new Error`, lo cual permite desplegar un nuevo error con un mensaje personalizado en su interior.

De esta forma, hemos logrado considerar ciertos elementos cruciales para poder empezar a manejar las excepciones dentro de nuestro código `async/await`.

VENTAJAS DE LAS PROMESAS POR SOBRE LOS CALLBACKS

Cuando se hace un uso masivo de los `callbacks`, podemos caer muy fácilmente en el **CALLBACK HELL**, que es una secuencia de llamadas produciendo código poco legible, por lo que se debe evitar.

```
a(function (resultadoDeA){
 b(resultadoDeA, function (resultadoDeB){
  c(resultadoDeB, function (resultadoDeC){
   d(resultadoDeC, function (resultadoDeD){
    e(resultadoDeD, function (resultadoDeE))
     f(resultadoDeE, function (resultadoDeF){
       console.log(resultadoDeF)
     })
    })
   })
  })
 })
});
```

Para no caer en este anti patrón, se recomienda el uso de Promesas.

```
async function ejemploCH(){
 try{
  const resultadoDeA = await a();
  const resultadoDeB = await b(resultadoDeA);
  const resultadoDeC = await c(resultadoDeB);
  const resultadoDeD = await d(resultadoDeC);
  const resultadoDeE = await e(resultadoDeD);
  const resultadoDeF = await f(resultadoDeE);
  console.log(resultadoDeF)
 }
}
```

VENTAJAS Y DESVENTAJAS DE ASYNC/AWAIT

VENTAJAS

- Es más fácil de leer al tener una sintaxis clara y simple.
- Para el manejo de errores podemos utilizar TRY/CATCH.

DESVENTAJAS

- Al igual que las Promesas, Async/Await no es compatible con navegadores más antiguos, por lo que se necesita transpilar el código a través de polyfills.