**Resumen de la Clase: Uso de async y await**

En esta clase se introduce el uso de las palabras clave async y await en JavaScript, que permiten manejar funciones asíncronas de una manera más sencilla y legible.

**Concepto de Promesas**

Al principio, el profesor explica cómo se utilizan las promesas para manejar operaciones asíncronas. Se crea una función que devuelve una promesa usando la siguiente sintaxis:

javascript

Copiar código

// Se crea una nueva promesa que simula una operación asíncrona

const getImagenPromesa = () => {

const promesa = new Promise((resolve, reject) => {

resolve("https://totalspirits.cl/en/");

});

return promesa;

};

// Uso de .then para obtener el resultado de la promesa

getImagenPromesa().then(console.log);

**Simplificación de Promesas**

Luego, muestra cómo simplificar este código usando una versión más corta de la promesa, eliminando el uso del reject y usando una única línea para resolver la promesa:

javascript

Copiar código

// Versión simplificada de la promesa

const getImagenPromesa = () =>

new Promise((resolve) => resolve("https://totalspirits.cl/en/"));

// Llamada a la promesa y muestra el resultado en consola

getImagenPromesa().then(console.log);

**Introducción de async y await**

El profesor menciona que podemos usar async para transformar una función en una que devuelve una promesa de forma implícita. Además, al usar await dentro de funciones async, podemos esperar el resultado de una promesa sin necesidad de usar .then(). Así es como se implementa el código:

javascript

Copiar código

// Función que devuelve una promesa de manera sencilla con async

const getImagen = async () => {

return "https://totalspirits.cl/en/";

};

// Llamada a la función y muestra el resultado en consola

getImagen().then(console.log);

En este caso, la palabra clave async hace que la función devuelva una promesa, y al usar await se puede manejar de manera más sencilla el resultado de las promesas.

**Uso de await para Trabajar con Fetch**

Más adelante, el profesor muestra cómo utilizar async y await para hacer una petición HTTP de manera más sencilla. Usando la API de Giphy, el código se ve de la siguiente manera:

javascript

Copiar código

// Función asincrónica que obtiene una imagen desde la API de Giphy

const getImagen = async () => {

try {

const apiKey = "6x7IRomHWVGl9GnyJxdmwgPmmlV4u41S";

// Espera la respuesta de la petición fetch

const resp = await fetch(

`https://api.giphy.com/v1/gifs/random?api\_key=${apiKey}`

);

// Espera que la respuesta se convierta en JSON

const { data } = await resp.json();

// Extrae la URL de la imagen

const { url } = data.images.original;

// Crea un elemento de imagen y lo agrega al documento

const img = document.createElement("img");

img.src = url;

document.body.append(img);

} catch (error) {

// Manejo de errores

console.error(error);

}

};

// Llamada a la función para mostrar la imagen

getImagen();

**Explicación Detallada**

1. **async**: Al declarar la función getImagen con async, la función automáticamente devuelve una promesa. Dentro de esta función, podemos usar await para esperar los resultados de las operaciones asíncronas.
2. **await**: Se utiliza para esperar el resultado de una promesa, como en la operación await fetch(). Esto hace que el código se ejecute de manera secuencial, como si fuera código síncrono, pero sin bloquear el hilo principal.
3. **fetch**: Se usa para realizar una petición HTTP a la API de Giphy, obteniendo un GIF aleatorio. El await garantiza que no se ejecuten otras líneas de código hasta que la respuesta de la petición sea recibida.
4. **Manejo de errores**: Dentro de un bloque try...catch, cualquier error en la ejecución de la función será capturado y mostrado en la consola.