Claro, vamos a desglosar esa parte del código para que sea más fácil de entender:

javascript

Copiar código

return new Promise((resolve, reject) => {

(true)

**1. new Promise((resolve, reject) => { ... })**

* **Promise**: Aquí estás creando una **nueva promesa** con el constructor Promise. Una promesa es un objeto en JavaScript que representa una operación asíncrona que puede completarse con éxito o fallar. Tiene dos posibles resultados:
  + **resolve**: Llamado cuando la operación es exitosa.
  + **reject**: Llamado cuando ocurre un error o la operación falla.

**2. (resolve, reject) => { ... }**

* Esta parte es una **función anónima** que se ejecuta cuando la promesa se crea. Recibe dos parámetros:
  + **resolve**: Función que llamas cuando quieres que la promesa se cumpla exitosamente.
  + **reject**: Función que llamas cuando quieres que la promesa se rechace, indicando que ocurrió un error.

**3. (true) ? ... : ...**

* Aquí estamos usando un **operador ternario**, que es una forma abreviada de escribir un if-else. La sintaxis básica es:

javascript

Copiar código

condición ? resultado\_si\_verdadero : resultado\_si\_falso

* + En este caso, la condición es (true), que siempre es verdadera, por lo que se ejecuta el lado del "resultado verdadero", es decir, el código que viene después del ?.

Entonces, cuando esta línea de código se ejecuta:

* **resolve('Async!!')** se ejecutará después de 2 segundos, cumpliendo la promesa exitosamente.
* Si la condición fuese falsa, se ejecutaría el reject, lo que significa que se rechazaría la promesa y arrojaría un error.

En resumen, esta línea está creando una promesa que, tras 2 segundos, se resolverá con el mensaje 'Async!!'.

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

? setTimeout(() => resolve('Async!!'), 2000)

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

Claro, desglosaré esta parte del código:

javascript

Copiar código

? setTimeout(() => resolve('Async!!'), 2000)

**Contexto:**

Esta es la parte del operador ternario que se ejecuta cuando la condición es verdadera. Vamos a verlo por partes.

**1. setTimeout(() => resolve('Async!!'), 2000)**

* **setTimeout**: Es una función nativa de JavaScript que se usa para ejecutar una función después de un período de tiempo específico. Su sintaxis básica es:

javascript

Copiar código

setTimeout(función, tiempo\_en\_milisegundos);

En este caso, la función a ejecutar es:

javascript

Copiar código

() => resolve('Async!!')

Y el tiempo de espera es de **2000 milisegundos (2 segundos)**.

**2. () => resolve('Async!!')**

* **Función flecha (() =>)**: Esta es una **función anónima** escrita en la sintaxis de función flecha. No tiene parámetros y se ejecuta después de los 2 segundos.
* **resolve('Async!!')**: Dentro de esta función, se llama a resolve con el argumento 'Async!!'. Esto significa que, después de los 2 segundos, la promesa se cumple (se resuelve) y el valor 'Async!!' es pasado como resultado de la promesa.

**3. 2000**

* **2000**: Este es el **tiempo de espera en milisegundos** (2 segundos) antes de que se ejecute la función que contiene resolve('Async!!'). Después de 2 segundos, la promesa se resolverá con el valor 'Async!!'.

**Resumen:**

* Si la condición en el operador ternario es verdadera, se ejecuta setTimeout, lo que significa que después de 2 segundos, se llamará a resolve('Async!!'), resolviendo la promesa con el valor 'Async!!'.

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

reject(new Error('Error!'));

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

Claro, vamos a desglosar esta línea de código:

javascript

Copiar código

reject(new Error('Error!'));

**1. reject()**

* **reject**: Es una función que forma parte del mecanismo de una promesa en JavaScript. Se usa para **indicar que la promesa ha fallado** o no ha sido cumplida correctamente.
* Llamar a reject() pasa la promesa al estado de "rechazada", lo que significa que algo salió mal en la operación asincrónica que se estaba ejecutando.

**2. new Error('Error!')**

* **new Error()**: Es una forma de **crear un objeto de error** en JavaScript. Este objeto describe lo que salió mal y puede llevar un mensaje personalizado. En este caso, se está creando un nuevo error con el mensaje 'Error!'.
* **Error** es un constructor en JavaScript que se utiliza para crear objetos de tipo error. Estos objetos pueden ser utilizados para describir y manejar errores de manera más clara.

**¿Qué hace esta línea en el código?**

* Si la promesa falla (cuando el operador ternario lo decide), se ejecuta reject(), indicando que la operación ha fallado.
* El argumento de reject es un objeto de error creado con new Error('Error!'). Esto significa que se está rechazando la promesa y el mensaje asociado al error será 'Error!'.

**Resumen:**

* **reject(new Error('Error!'))** indica que, si algo sale mal en la ejecución del código asincrónico, la promesa será rechazada y se enviará un mensaje de error ('Error!') describiendo lo que falló.

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

const anotherFn = async () => {

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

Claro, vamos a desglosar esta línea de código:

javascript

Copiar código

const anotherFn = async () => {

**1. const anotherFn**

* **const**: Es una palabra clave en JavaScript que se utiliza para declarar una **constante**. En este caso, estamos declarando una función que no cambiará de valor.
* **anotherFn**: Es el nombre de la constante. En este caso, se trata de una función que está siendo asignada a esta constante. El nombre de la función es arbitrario y puede ser cambiado por cualquier otro, como myFunction.

**2. async () =>**

* **async**: Es una palabra clave en JavaScript que se utiliza para definir una **función asíncrona**. Las funciones asíncronas permiten que el código dentro de ellas use await para manejar operaciones que toman tiempo (como las promesas) sin bloquear la ejecución del resto del código.
* **() =>**: Es la **sintaxis de una función de flecha**. Las funciones de flecha son una forma más concisa de escribir funciones en JavaScript, especialmente cuando no se necesita utilizar el this dentro de la función.

**¿Qué hace esta línea?**

* Esta línea declara una **función asíncrona** llamada anotherFn, utilizando la sintaxis de función de flecha.
* El uso de async indica que esta función contendrá código asíncrono, lo que significa que podrá utilizar await para esperar a que se resuelvan promesas dentro de ella.

**Resumen:**

* **const anotherFn = async () => {** es una declaración de una función asíncrona que permitirá manejar código asíncrono dentro de ella.

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

const somethig = await fnAsync();

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

Vamos a desglosar esta línea de código:

javascript

Copiar código

const somethig = await fnAsync();

**1. const somethig**

* **const**: Es una palabra clave en JavaScript utilizada para declarar una constante. En este caso, se está creando una variable llamada somethig (parece un error tipográfico, probablemente quisiste decir something).
* **somethig**: Es el nombre de la variable que almacenará el valor resultante de la ejecución de fnAsync().

**2. await fnAsync()**

* **await**: Es una palabra clave que se utiliza dentro de funciones asíncronas (aquellas definidas con async). Lo que hace await es esperar a que una **promesa** se resuelva. En lugar de seguir ejecutando el código inmediatamente, JavaScript espera a que la promesa retorne un valor.
* **fnAsync()**: Es una llamada a la función fnAsync(), que es una función que devuelve una promesa. En este caso, fnAsync() se resuelve con el valor 'Async!!' después de 2 segundos, gracias al uso de setTimeout.

**¿Qué hace esta línea?**

* **Primero**, la función fnAsync() es invocada y retorna una promesa.
* **Luego**, el await espera a que esa promesa sea resuelta.
  + Si la promesa se resuelve correctamente, el valor resultante (en este caso 'Async!!') es asignado a la variable somethig.
  + Si la promesa es rechazada, se lanzaría un error (aunque no está manejado aquí con un try/catch).

**Resumen:**

* La línea **const somethig = await fnAsync();**:
  + Llama a la función fnAsync(), que devuelve una promesa.
  + Utiliza await para esperar a que esa promesa se resuelva.
  + Una vez que la promesa se resuelve, asigna el valor resultante a la constante somethig.

En este caso, después de 2 segundos, la variable somethig contendrá el valor 'Async!!'.

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

let xhttp = new XMLHttpRequest();

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓