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

import fetch from 'node-fetch'

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

Vamos a desglosar la línea de código import fetch from 'node-fetch'; en detalle:

### import fetch from 'node-fetch';

#### 1. import

* **¿Qué es?**: import es una palabra clave de JavaScript que se usa para traer o cargar funciones, objetos o variables que han sido exportadas desde otro archivo o módulo.
* **¿Cómo funciona?**: Cuando se utiliza import, se está diciendo que queremos acceder a algo (en este caso, fetch) desde otro archivo o módulo externo para poder usarlo en el archivo actual.

#### 2. fetch

* **¿Qué es?**: fetch es el nombre de la función que estamos importando. En este caso, viene del módulo node-fetch.
* **¿Qué hace?**: La función fetch se utiliza para hacer solicitudes HTTP (como obtener datos de una API). Básicamente, envía una solicitud a una URL y devuelve una promesa que se resuelve con la respuesta.

#### 3. from 'node-fetch'

* **¿Qué es?**: La parte from 'node-fetch' indica de dónde estamos importando la función fetch.
* **node-fetch** es un paquete de npm (Node.js Package Manager) que proporciona una implementación de la API de fetch en entornos de Node.js. Esto es necesario porque fetch no está disponible de forma nativa en Node.js, aunque sí lo está en navegadores.

### En conjunto:

* **Lo que hace esta línea**: Importa la función fetch del paquete node-fetch para que puedas usarla en tu código. Esta función te permite hacer solicitudes HTTP a una URL, lo que es común al interactuar con APIs.

### Analogía:

Piensa en import como cuando traes herramientas de una caja de herramientas que tiene otra persona. En este caso, la caja de herramientas se llama node-fetch, y la herramienta específica que estás tomando es la función fetch. Ahora puedes usar esa herramienta (función fetch) en tu propio trabajo (tu archivo de código) para obtener datos de una API.

### Resumen:

* import es la palabra clave para traer algo de otro lugar.
* fetch es la función específica que estás trayendo.
* from 'node-fetch' te dice de qué módulo o paquete estás trayendo la función fetch.

¿Te quedó claro el funcionamiento?

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

import fetch from 'node-fetch'

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

Con gusto! Vamos a desglosar esta línea de código:

js

Copiar código

function fetchData(urlApi) {

### Desglose de la línea:

1. **function**:
   * Esta palabra clave de JavaScript se utiliza para declarar una **función**. Una función es un bloque de código que realiza una tarea específica cuando es llamada o invocada.
2. **fetchData**:
   * Este es el **nombre de la función**. En este caso, la función se llama fetchData. El nombre es arbitrario, pero en este caso describe lo que hace la función: "obtener datos". Es importante elegir nombres descriptivos que indiquen qué hace la función.
3. **(urlApi)**:
   * Lo que está dentro de los paréntesis es un **parámetro**. En este caso, el parámetro es urlApi, que es una variable que actuará como un "marcador de posición" para cualquier valor que se pase cuando la función sea invocada.
   * **urlApi** es el nombre del parámetro y representa la URL de una API desde la cual se pretende obtener datos. Cuando alguien llame a la función fetchData, deberá proporcionar un argumento para este parámetro, que generalmente será una URL en forma de cadena de texto (por ejemplo, 'https://api.escuelajs.co/api/v1/products').
4. **{**:
   * La llave de apertura { indica el comienzo del **bloque de código** de la función. Todo el código que define lo que hace la función irá dentro de estas llaves. Cuando se invoca fetchData, todo lo que esté dentro de este bloque de código se ejecutará.

### ¿Qué hace esta línea en conjunto?

Esta línea declara una función llamada fetchData que recibirá un argumento (que será una URL) y dentro de esta función, se realizará alguna acción (probablemente relacionada con la obtención de datos desde esa URL).

Es como crear una receta de cocina, donde la función es la receta en sí, el parámetro es un ingrediente que necesitas para la receta, y el bloque de código dentro de la función es el conjunto de instrucciones para preparar el platillo.

### Ejemplo de uso:

Si más adelante llamamos a esta función así:

js

Copiar código

fetchData('https://api.escuelajs.co/api/v1/products');

* urlApi en la función tomará el valor 'https://api.escuelajs.co/api/v1/products', y la función ejecutará su bloque de código usando esa URL para hacer lo que esté diseñado para hacer (en este caso, probablemente obtener datos de la API).

Espero que esta explicación te sea clara y útil. ¡Déjame saber si tienes más preguntas!

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

return fetch(urlApi);

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

Vamos a desglosar esta línea de código:

js

Copiar código

return fetch(urlApi);

### Desglose de la línea:

1. **return**:
   * La palabra clave return se utiliza en una función para **devolver** un valor o resultado. Cuando el intérprete de JavaScript encuentra un return, la función detiene su ejecución y devuelve el valor que sigue a return al lugar donde fue invocada la función.
   * En este caso, está devolviendo el resultado de la llamada a la función fetch(urlApi).
2. **fetch(urlApi)**:
   * **fetch** es un método nativo de JavaScript (disponible en navegadores modernos y entornos como Node.js) que se utiliza para realizar **solicitudes HTTP**. Este método permite obtener recursos (datos) de un servidor. Generalmente, fetch se usa para hacer solicitudes GET (para obtener datos), pero también puede hacer otras como POST, PUT, etc.
   * **urlApi** es el argumento que le pasamos al método fetch. En este caso, urlApi es una cadena de texto que contiene la **URL** del recurso (API) que queremos obtener. El valor de urlApi fue recibido como un parámetro de la función fetchData, lo cual permite que fetch haga la solicitud HTTP a esa URL.
3. **Resultado de fetch(urlApi)**:
   * **fetch devuelve una promesa**, lo que significa que no obtiene el resultado de inmediato. En cambio, promete que una vez que la solicitud HTTP haya sido procesada (éxito o error), devolverá un objeto de respuesta que contiene los datos solicitados o un error.
   * **¿Por qué devolver la promesa?**: Al utilizar return junto con fetch, estamos devolviendo esa promesa a quien haya llamado a la función fetchData. Esto permite que quien invoque fetchData pueda continuar trabajando con esa promesa, usando métodos como .then() o .catch() para manejar los resultados de la solicitud HTTP.

### ¿Qué hace esta línea en conjunto?

Esta línea dice que cuando alguien llama a la función fetchData y le pasa una URL (urlApi), la función hará una solicitud HTTP utilizando fetch para obtener datos desde esa URL. El resultado de esa solicitud (la promesa devuelta por fetch) es devuelto (con return) por la función.

En otras palabras, esta línea de código:

1. Realiza una solicitud HTTP a la URL proporcionada.
2. Devuelve una **promesa** para que el código que llame a la función pueda manejar la respuesta o el error de esa solicitud.

### Ejemplo de cómo se usaría:

js

Copiar código

fetchData('https://api.escuelajs.co/api/v1/products')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.error(error));

En este ejemplo:

* Llamamos a fetchData con una URL.
* fetchData devuelve la promesa de fetch.
* Luego usamos .then() para manejar la respuesta una vez que se complete la solicitud.
* Si hay un error, usamos .catch() para manejarlo.

Espero que esta explicación te aclare el funcionamiento. Si tienes más preguntas, ¡no dudes en preguntar!

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

fetchData(`${API}/products`)

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

Claro, vamos a desglosar esta línea de código:

js

Copiar código

fetchData(`${API}/products`)

**Desglose de la línea:**

1. **fetchData**:
   * Es el nombre de la función que se ha definido previamente en el código. La función fetchData toma como parámetro una **URL** (urlApi), realiza una solicitud HTTP utilizando fetch, y devuelve una **promesa** que contiene el resultado de esa solicitud.
2. **${API}/products**:
   * Este es el argumento que se le pasa a la función fetchData. Es una **plantilla de cadena** o **template literal** en JavaScript. Se usan las comillas invertidas (`) para crear una cadena de texto dinámica.
   * **${API}** es una **interpolación** que inserta el valor de la variable API en esa posición dentro de la cadena. En este caso, se asume que API tiene un valor que representa una URL base, como por ejemplo:

js

Copiar código

const API = 'https://api.escuelajs.co/api/v1';

* + Entonces, **${API}/products** construye una URL completa al concatenar la variable API con "/products", lo que resultaría en:

rust

Copiar código

'https://api.escuelajs.co/api/v1/products'

Esta URL se está utilizando para hacer una solicitud a un recurso específico (en este caso, una lista de productos).

1. **Función invocada**:
   * Cuando escribes fetchData(${API}/products), estás llamando a la función fetchData y pasándole la URL generada como argumento. La función entonces utilizará esta URL para hacer una solicitud HTTP a la API para obtener datos relacionados con productos.
2. **Lo que hace**:
   * La función fetchData enviará una solicitud HTTP GET a la URL 'https://api.escuelajs.co/api/v1/products'. Dado que fetchData devuelve una promesa, el código que llame a esta función (por ejemplo, con .then()) puede manejar la respuesta de la solicitud cuando se complete.

**Ejemplo de cómo funciona:**

js

Copiar código

const API = 'https://api.escuelajs.co/api/v1';

fetchData(`${API}/products`)

.then(response => response.json()) // Manejar la respuesta y convertirla en JSON

.then(data => console.log(data)) // Mostrar los datos obtenidos

.catch(error => console.error(error)); // Manejar cualquier error que ocurra

**¿Qué hace esta línea en conjunto?**

Esta línea de código hace lo siguiente:

1. Llama a la función fetchData para hacer una solicitud HTTP a la API en la URL 'https://api.escuelajs.co/api/v1/products'.
2. fetchData devuelve una promesa que, cuando se resuelva, contendrá los datos de los productos desde esa API.
3. La promesa devuelta puede ser manejada usando métodos como .then() para obtener los datos o .catch() para manejar errores si la solicitud falla.

Espero que esta explicación te haya aclarado cómo funciona esta línea. ¡Déjame saber si tienes más preguntas!

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

import fetch from 'node-fetch'

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

Claro, vamos a desglosar esta línea:

js

Copiar código

.then(response => response.json())

**Desglose de la línea:**

1. **.then()**:
   * **.then()** es un método que se utiliza con **promesas** en JavaScript. Se ejecuta cuando la promesa previa se resuelve con éxito. En este caso, la promesa anterior es la llamada a la función fetchData, que devuelve una promesa que contiene una **respuesta HTTP**.
2. **response**:
   * Dentro de .then(), el valor que se pasa es el resultado de la promesa resuelta. Aquí, response representa el **objeto de respuesta HTTP** devuelto por la función fetch() dentro de fetchData. Este objeto contiene información sobre la solicitud HTTP, incluyendo los datos, el estado de la respuesta, encabezados, etc.
3. **=> (función de flecha)**:
   * Este símbolo define una **función de flecha**. En este caso, es una función corta que recibe un parámetro (response) y devuelve algo. Aquí se está utilizando para transformar o procesar los datos de la respuesta.
4. **response.json()**:
   * El objeto response tiene varios métodos, uno de los cuales es .json(). Este método se utiliza para **convertir** la respuesta HTTP (que está en formato **JSON**, o texto en formato de objeto) en un **objeto JavaScript**.
   * El formato JSON es un estándar para el intercambio de datos que representa objetos y matrices de manera textual, y .json() facilita convertirlo a un formato que JavaScript puede utilizar fácilmente.
   * **Nota**: .json() también devuelve una **promesa**, porque la conversión de datos podría llevar algún tiempo, especialmente si los datos son grandes o complejos. Eso es porque procesar la respuesta en JSON puede no ser instantáneo.

**Lo que hace en conjunto:**

* Cuando la promesa de fetchData se resuelve (es decir, cuando se completa la solicitud HTTP y llega la respuesta), el objeto de la respuesta pasa a través de .then().
* Luego, response.json() convierte esa respuesta en **JSON legible por JavaScript** (un objeto o arreglo).
* Como .json() devuelve otra promesa, la cadena de promesas continúa. El siguiente .then() manejará el objeto ya convertido.

**Ejemplo paso a paso:**

js

Copiar código

fetchData(`${API}/products`)

.then(response => response.json()) // Convierte la respuesta en un objeto JSON

.then(data => console.log(data)) // `data` es ahora el JSON que contiene los productos

.catch(error => console.error(error)); // Maneja cualquier error

1. **Primera promesa (fetchData)**: Se hace una solicitud a la API.
2. **Primer .then() (response.json())**: La respuesta HTTP de la solicitud se convierte de texto JSON a un objeto JavaScript.
3. **Segundo .then()**: Ahora se puede trabajar con los datos de la API que fueron convertidos a JSON.
4. **.catch()**: Maneja cualquier error en el proceso.

**Analogía:**

Imagina que pides un paquete (la solicitud HTTP) y cuando te lo entregan (la respuesta HTTP), está envuelto en varias capas de plástico. **.json()** es como quitar ese plástico y dejar al descubierto el contenido que realmente quieres usar (el objeto JSON que contiene los datos).

Espero que esta explicación haya sido clara. Si necesitas más detalles o ejemplos, ¡dímelo!

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

.then(products => {

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

¡Claro! Vamos a desglosar la línea de código:

javascript

Copiar código

.then(products => {

**Contexto**

Este fragmento de código es parte de una cadena de promesas (Promise). La función then() se ejecuta cuando la promesa anterior se ha resuelto con éxito, y el valor devuelto por esa promesa es pasado a la función que se encuentra dentro de then().

**Explicación paso a paso:**

1. **then()**:
   * then es un método que pertenece a las promesas en JavaScript. Se ejecuta cuando la promesa anterior se resuelve correctamente. Este método toma una función como argumento, que se encargará de procesar el valor que retorna la promesa.
2. **products => {**:
   * Este es el comienzo de una **función flecha** (arrow function), que es la función que se ejecuta dentro del then().
   * products es el **argumento** que recibe la función. En este caso, será el resultado que devuelve la promesa anterior. Es decir, la promesa anterior devolvió un array de productos (probablemente en formato JSON después de ser procesado con response.json()).
3. **Procesamiento dentro del bloque**:
   * El bloque de código que sigue dentro de las llaves { se ejecutará con el valor products. Aquí puedes hacer lo que quieras con los productos obtenidos de la API, como por ejemplo, mostrarlos en la consola o pasarlos a otra función para más procesamiento.

Un ejemplo de lo que sigue podría ser algo como:

javascript

Copiar código

console.log(products);

Aquí se imprimirían los productos en la consola.

**Resumen:**

* then(products => { está diciendo: "Cuando la promesa anterior se resuelva con éxito, ejecuta esta función, usando el valor de la promesa resuelta como argumento llamado products".

Es como recibir el paquete (la respuesta de la API) y luego abrirlo (procesar los datos en la función flecha).

Espero que esto te haya quedado claro. Si tienes más preguntas, ¡solo dime! 😊

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

return fetchData(`${API}/products/${products[0].id}`);

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

¡Vamos a desglosar esta línea paso a paso!

javascript

Copiar código

return fetchData(`${API}/products/${products[0].id}`);

**Contexto**

Esta línea está dentro de una cadena de promesas y su propósito es realizar una nueva solicitud a la API utilizando un producto específico obtenido de una lista anterior. Vamos a ver cómo funciona esto en detalle.

**Explicación paso a paso:**

1. **return**:
   * La palabra clave return se utiliza para devolver un valor o una expresión desde una función. En este caso, está devolviendo el resultado de la llamada a fetchData(), que es otra promesa.
   * Al usar return en este punto dentro de un .then(), la nueva promesa se encadenará con la anterior. Esto permite continuar con el flujo asíncrono de la cadena de promesas.
2. **fetchData(...)**:
   * fetchData es la función que definiste anteriormente, y se encarga de realizar una solicitud HTTP a la API que especificaste.
   * Cuando llamas a fetchData(), envías una solicitud a una URL de la API y devuelves una promesa con la respuesta.
3. **${API}/products/${products[0].id}**:
   * **${API}**: Esto es una **template literal** que incluye la variable API en la cadena. API contiene la URL base de la API que estás consultando, en este caso algo como 'https://api.escuelajs.co/api/v1'.
   * **/products/${products[0].id}**: Aquí estás accediendo al primer producto de la lista de productos que recibiste en la promesa anterior. products[0] es el primer objeto en el array products, y con .id obtienes el **id** de ese primer producto.
   * Por lo tanto, estás construyendo una URL específica para ese producto, como por ejemplo: 'https://api.escuelajs.co/api/v1/products/1', si el ID del producto es 1.
4. **Flujo completo**:
   * Lo que estás haciendo aquí es hacer una nueva solicitud para obtener los detalles de un producto específico, en este caso, el primer producto de la lista que obtuviste en la promesa anterior.

**Resumen:**

* **Objetivo**: Realizas una nueva solicitud para obtener detalles específicos del primer producto (products[0]) en la lista de productos que recibiste en la promesa anterior.
* **Encadenamiento**: Al devolver fetchData, esta promesa se encadena con las promesas anteriores, lo que permite que el flujo de la cadena continúe ejecutándose de manera asíncrona.

Es como si primero pidieras una lista de productos, eligieras el primero, y luego hicieras una nueva consulta para obtener más detalles sobre ese producto específico.

Espero que esto haya quedado claro. ¡No dudes en preguntarme si necesitas más detalles! 😊

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

return fetchData(`${API}/categories/${product.category.id}`);

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

Vamos a desglosar esta línea paso a paso!

javascript

Copiar código

return fetchData(`${API}/categories/${product.category.id}`);

**Contexto**

Esta línea está dentro de la cadena de promesas y sigue la lógica de consultar más información sobre un producto. Específicamente, después de obtener un producto, ahora realizas una solicitud para obtener los detalles de la categoría a la que pertenece ese producto.

**Explicación paso a paso:**

1. **return**:
   * La palabra clave return se utiliza para devolver el resultado de fetchData(). En el contexto de una cadena de promesas, return permite que la nueva promesa (la solicitud de la categoría) se encadene con las anteriores. Esto asegura que el flujo de la operación asíncrona continúe.
2. **fetchData(...)**:
   * La función fetchData realiza una solicitud HTTP a una URL que le pasas como argumento. Esta función devuelve una **promesa** con el resultado de esa solicitud.
   * Aquí, se está utilizando fetchData para realizar una nueva consulta a la API.
3. **${API}/categories/${product.category.id}**:
   * Esta es una **template literal** que se usa para construir dinámicamente una URL basada en la información obtenida previamente.
   * **${API}**: Aquí estás reutilizando la variable API, que contiene la URL base de la API, en este caso algo como 'https://api.escuelajs.co/api/v1'.
   * **/categories/${product.category.id}**:
     + **product.category.id**: product es el objeto que obtuviste en la promesa anterior. Este objeto tiene una propiedad category, que a su vez tiene una propiedad id. Así que aquí estás accediendo al **ID** de la categoría del producto.
     + La ruta completa se construye como, por ejemplo, 'https://api.escuelajs.co/api/v1/categories/5', si el ID de la categoría es 5.
4. **Flujo completo**:
   * Al realizar esta solicitud, estás buscando obtener más detalles sobre la categoría a la que pertenece el producto que ya obtuviste. Después de obtener el producto, ahora haces una solicitud para obtener los detalles de la categoría utilizando su ID.

**Resumen:**

* **Objetivo**: Realizas una nueva solicitud a la API para obtener detalles de la categoría de un producto específico (product.category.id).
* **Encadenamiento**: El return permite que la promesa devuelta por fetchData se encadene con las promesas anteriores, manteniendo el flujo asíncrono.
* **URL dinámica**: Utilizas una **template literal** para construir una URL dinámica, que incluye el ID de la categoría obtenida del producto anterior.

Es como si primero obtuvieras información sobre un producto, y luego, con esa información, pidieras detalles sobre la categoría a la que pertenece el producto.

¡Espero que esta explicación te haya resultado útil! ¿Te gustaría que siga explicando alguna otra parte? 😊

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

catch(err => console.log(err))

### .finally(() => console.log('Finally'));

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

Claro, te explicaré cada una de esas líneas de código:

### .catch(err => console.log(err))

1. **.catch()**: Este método es parte de las promesas en JavaScript y se utiliza para manejar errores. Si alguna parte de la cadena de promesas genera un error, se ejecutará la función que se pasa dentro de catch(). Esto permite capturar y manejar cualquier fallo que ocurra, ya sea en la solicitud HTTP, en el procesamiento de los datos, o en cualquier otro paso.
2. **err => console.log(err)**: Esta es una función flecha que recibe un parámetro, en este caso err, que contiene el error. Luego, utiliza console.log(err) para imprimir el error en la consola. En resumen, si algo sale mal durante la ejecución de las promesas anteriores, se ejecutará esta línea y mostrará el error en la consola.

**Ejemplo**:

* + Si la solicitud a la API falla por alguna razón (por ejemplo, no se encuentra la URL o hay un problema con la red), el error será capturado por .catch(), y luego err contendrá la información del error que será registrada en la consola.

### .finally(() => console.log('Finally'));

1. **.finally()**: Este método se ejecuta independientemente de si la promesa se resolvió correctamente (.then()) o si falló (.catch()). Es ideal para realizar acciones que siempre deben ocurrir, como limpiar recursos, detener indicadores de carga, etc. No recibe ningún valor ni depende del resultado de la promesa.
2. **() => console.log('Finally')**: Esta es otra función flecha que no toma parámetros. Lo que hace es simplemente imprimir el mensaje 'Finally' en la consola cuando se ejecuta. El uso de finally() garantiza que este mensaje siempre se mostrará, sin importar si la promesa fue exitosa o fallida.

**Ejemplo**:

* + Después de que todas las solicitudes de datos hayan terminado (ya sea con éxito o error), se ejecutará este bloque de código y el mensaje 'Finally' será impreso en la consola.

### En resumen:

* **.catch(err => console.log(err))**: Se encarga de capturar y manejar errores, mostrando el error en la consola si ocurre.
* **.finally(() => console.log('Finally'))**: Se ejecuta al final de la cadena de promesas, independientemente de si hubo éxito o error, imprimiendo 'Finally' en la consola.

Esto asegura que siempre se registre un mensaje al finalizar el proceso completo de las promesas, y cualquier error sea manejado correctamente.

¿Te ha quedado claro o hay alguna parte que necesites más detalles?

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

import fetch from 'node-fetch'

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