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

**function postData(urlApi, data) {**

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

Vamos a desglosar esta línea de código:

**function postData(urlApi, data) {**

1. **function**:
   * En JavaScript, la palabra clave function se utiliza para declarar una función. Las funciones son bloques de código que realizan tareas específicas. Al declarar una función, estás creando un conjunto de instrucciones que se puede ejecutar cuando se invoca o llama a la función.
2. **postData**:
   * Este es el **nombre de la función**. En este caso, el programador ha decidido llamarla postData, lo que sugiere que la función probablemente enviará (o "posteará") datos a una API (interfaz de programación de aplicaciones).
3. **(urlApi, data)**:
   * Estas son las **entradas o parámetros** de la función. Una función puede recibir uno o más parámetros que actúan como variables dentro de la función.
     + urlApi: Es el primer parámetro, que representa la URL a la que se quiere enviar la solicitud. Al llamar a la función, el valor de esta variable será una URL, por ejemplo, https://api.escuelajs.co/api/v1/products.
     + data: Es el segundo parámetro, que representa los datos que se van a enviar. Estos datos serán transformados en JSON antes de enviarse a la API.
4. **{**:
   * Este símbolo marca el **inicio del bloque de código** que define el comportamiento de la función. Todo lo que esté entre este { y el correspondiente } al final, será ejecutado cuando la función postData sea llamada.

**Resumen:**

Esta línea declara una función llamada postData que acepta dos parámetros: urlApi y data. La función está diseñada para trabajar con estos dos valores y realizar alguna operación, que se definirá en las líneas de código siguientes. En este caso, como sugiere el nombre, parece que va a enviar datos a una API, utilizando la URL y los datos proporcionados.

¿Te quedó claro hasta aquí?

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

### const response = fetch(urlApi, { method: 'POST', mode: 'cors', credentials: 'same-origin', headers: { 'Content-Type': 'application/json' } })

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

¡Vamos a desglosar esta línea de código!

### const response = fetch(urlApi, { method: 'POST', mode: 'cors', credentials: 'same-origin', headers: { 'Content-Type': 'application/json' } })

#### 1. const:

* const es una palabra clave en JavaScript que se usa para declarar una **variable constante**. Una vez que se asigna un valor a una variable con const, no se puede reasignar. En este caso, la variable se llama response.

#### 2. response:

* Este es el **nombre de la variable**. Se usa para almacenar el valor devuelto por la función fetch(). Es decir, response contendrá el resultado de la solicitud HTTP que se está realizando.

#### 3. fetch(urlApi, {...}):

* **fetch()** es una función incorporada en JavaScript que se utiliza para realizar solicitudes HTTP (en este caso, una solicitud para enviar datos a una API). La función devuelve una **promesa**, lo que significa que manejará la respuesta de manera asíncrona (es decir, la respuesta puede tardar en llegar y no bloqueará el flujo de ejecución del código mientras espera).
* El primer parámetro que se pasa a fetch() es **urlApi**, que es la **URL** a la que se enviará la solicitud. En este caso, la URL se pasará cuando la función postData sea invocada.
* El segundo parámetro es un **objeto de configuración** que le dice a fetch() cómo debe realizar la solicitud.

#### 4. method: 'POST':

* Esta es una propiedad del objeto de configuración que le dice a fetch() qué tipo de solicitud HTTP debe hacer.
* 'POST' es uno de los **métodos HTTP** más comunes y se usa cuando se quiere **enviar datos** al servidor. En este caso, la función postData está enviando datos a la URL proporcionada.

#### 5. mode: 'cors':

* Esta propiedad indica cómo se gestionan las solicitudes a diferentes dominios (cross-origin).
* **'cors'** (Cross-Origin Resource Sharing) es una configuración que permite que un sitio web solicite recursos desde un dominio diferente al que sirvió la página web. Esta opción es común cuando se trabaja con APIs públicas o de terceros.

#### 6. credentials: 'same-origin':

* Esta propiedad controla si se deben enviar las **cookies** o credenciales de autenticación junto con la solicitud.
* **'same-origin'** significa que las credenciales solo se enviarán si la solicitud es al mismo dominio de donde proviene la página. Es una medida de seguridad que protege las credenciales cuando se hacen solicitudes a diferentes orígenes.

#### 7. headers: { 'Content-Type': 'application/json' }:

* **headers** son metadatos que se envían con la solicitud HTTP para proporcionar información adicional al servidor.
* **'Content-Type': 'application/json'** especifica que el cuerpo de la solicitud estará en formato JSON (JavaScript Object Notation). Es importante cuando se están enviando datos al servidor, ya que este necesita saber en qué formato están los datos para procesarlos correctamente.

#### 8. { **y** }:

* Los corchetes {} marcan el inicio y el final del **objeto de configuración** que contiene todas las propiedades que le dicen a fetch() cómo hacer la solicitud.

### Resumen:

Esta línea de código usa fetch() para enviar una solicitud HTTP tipo POST a la URL proporcionada (representada por urlApi). Configura la solicitud para que trabaje en modo CORS, y para que envíe los datos como JSON en el cuerpo de la solicitud. Los resultados de la solicitud se almacenan en la constante response.

¿Te quedó claro hasta aquí?

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

### body: JSON.stringify(data)

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

Vamos a desglosar esta parte del código:

### body: JSON.stringify(data)

#### 1. body:

* **body** es una propiedad del objeto de configuración que le dice a la función fetch() qué datos se deben enviar en el **cuerpo de la solicitud** HTTP.
* En una solicitud POST (como es el caso), el **cuerpo** contiene los datos que estás enviando al servidor. Es como el "contenido" que quieres enviar, similar a llenar un formulario y enviarlo.

#### 2. JSON.stringify(data):

* **JSON.stringify()** es un método en JavaScript que **convierte** un objeto JavaScript (en este caso, el objeto data) en una **cadena de texto** en formato JSON.
* JSON (JavaScript Object Notation) es un formato de datos que se usa comúnmente para enviar y recibir datos en aplicaciones web. Los servidores suelen entender los datos en formato JSON, por lo que, antes de enviar el objeto data, es necesario convertirlo en una cadena de texto en este formato.

#### Ejemplo del uso de JSON.stringify:

* Si el objeto data es:

js

Copiar código

const data = {

title: "212",

price: 212,

description: "A description",

categoryId: 1,

images: ["https://placeimg.com/640/480/any"]

};

* **JSON.stringify(data)** lo convertirá en una cadena JSON como esta:

json

Copiar código

'{"title":"212","price":212,"description":"A description","categoryId":1,"images":["https://placeimg.com/640/480/any"]}'

#### 3. ;:

* El punto y coma ; marca el final de la declaración. En este caso, indica el fin de la línea que asigna el cuerpo de la solicitud POST.

### ¿Qué hace esta parte del código?

Este fragmento de código está preparando el **cuerpo** de la solicitud HTTP para ser enviado al servidor. Está tomando los datos que quieres enviar (data) y los está convirtiendo en una **cadena JSON** que el servidor puede entender. Después, este cuerpo (con los datos convertidos) se incluye en la solicitud HTTP POST como parte de la propiedad body.

### Resumen:

* La propiedad body contiene los datos que se enviarán en la solicitud.
* **JSON.stringify(data)** convierte el objeto JavaScript en una cadena JSON para que pueda ser enviado correctamente en la solicitud HTTP.

¿Te quedó claro este concepto?

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

postData(${API}/products, data)

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

### Resumen

La línea postData(${API}/products, data) llama a la función postData y le pasa dos argumentos:

* La URL donde enviar los datos (https://api.escuelajs.co/api/v1/products).
* El objeto data, que contiene la información del nuevo producto a enviar.

En resumen, esta línea inicia el proceso de enviar los datos del producto al servidor mediante una solicitud HTTP POST.

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

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

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

¡Claro! Vamos a desglosar esta línea de código:

javascript

Copiar código

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

### Explicación

1. **.then(...)**:
   * .then() es un método que se utiliza en una promesa (como la que devuelve fetch).
   * Indica lo que se debe hacer cuando la promesa se resuelve correctamente, es decir, cuando la solicitud POST que hiciste previamente con postData recibe una respuesta del servidor.
2. **response**:
   * El parámetro response dentro de la función flecha (response => ...) representa la respuesta que el servidor envía después de que la solicitud POST se completa.
   * Este response es un objeto que contiene información sobre la respuesta, como el estado (status), los encabezados, y el cuerpo (que en este caso es un JSON).
3. **response.json()**:
   * La propiedad json() es un método del objeto response.
   * Lo que hace es convertir el cuerpo de la respuesta (response body) de la solicitud HTTP en un objeto JavaScript, ya que el cuerpo de la respuesta viene en formato JSON.
   * Cuando llamas a response.json(), estás extrayendo los datos en formato JSON y transformándolos en un objeto para poder trabajar con ellos fácilmente en tu código.
4. **Promesa devuelta**:
   * response.json() también devuelve una promesa, porque la conversión de JSON puede ser asíncrona. Por eso, puedes encadenar otro .then() después para trabajar con los datos ya convertidos.

### Resumen

La línea then(response => response.json()) hace lo siguiente:

1. Cuando se resuelve la promesa de postData, recibe la respuesta del servidor (response).
2. Luego, convierte el cuerpo de esa respuesta de formato JSON a un objeto JavaScript usando response.json(), para que puedas acceder a sus propiedades en los siguientes pasos.

Así, el JSON que contiene los datos del servidor se transforma en un formato que puedes utilizar en tu aplicación.

¿Te queda claro cómo funciona esta parte?

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

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

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

Vamos a desglosar esta línea!

javascript

Copiar código

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

### Explicación

1. **.then(...)**:
   * El método .then() se utiliza para manejar el resultado de una promesa. En este caso, la promesa es la conversión del JSON de la respuesta que hiciste en el .then anterior (response.json()).
   * Esta parte del código se ejecutará después de que la promesa anterior (la conversión de la respuesta a JSON) se haya resuelto con éxito.
2. **data**:
   * data es el nombre del parámetro que recibe los datos que han sido convertidos a un objeto JavaScript por la llamada a response.json() en el .then anterior.
   * Este objeto data contiene los detalles de la respuesta enviada por el servidor, que puede incluir la información del producto que acabas de crear.
3. **console.log(data)**:
   * console.log() es una función de JavaScript que imprime información en la consola del navegador o del entorno en el que estás ejecutando el código (como Node.js).
   * Al llamar a console.log(data), lo que haces es mostrar el contenido de data (el objeto con la respuesta del servidor) en la consola para que puedas verlo.
   * Esta es una técnica muy común para verificar qué datos se están recibiendo del servidor y asegurarse de que todo funciona correctamente.

### Resumen

Esta línea de código:

1. Recibe los datos ya convertidos a un objeto JavaScript (en este caso, el producto que acabas de crear).
2. Luego, los imprime en la consola con console.log(data), permitiéndote visualizar el contenido de la respuesta.

Es una forma útil de depurar y verificar que los datos que obtuviste del servidor son correctos. ¿Te quedó claro cómo funciona esta parte?

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

import fetch from 'node-fetch'

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