**API Fetch**

Contenido

[1. Introducción 1](#_Toc189731886)

[**Características principales de Fetch:** 2](#_Toc189731887)

[**Ejemplo básico de uso:** 2](#_Toc189731888)

[2. El objeto Response en el API Fetch 2](#_Toc189731889)

[**Propiedades del objeto Response** 2](#_Toc189731890)

[**Métodos del objeto Response** 3](#_Toc189731891)

[**Manejo de errores con Response** 4](#_Toc189731892)

[**Conclusión** 4](#_Toc189731893)

[3. Métodos de la petición 4](#_Toc189731894)

[4. Método GET en el API Fetch 5](#_Toc189731895)

[Sintaxis básica 5](#_Toc189731896)

[Ejemplo con parámetros en la URL 5](#_Toc189731897)

[Ejemplo con cabeceras personalizadas 5](#_Toc189731898)

[Manejo de errores en una petición GET 6](#_Toc189731899)

[Conclusión 6](#_Toc189731900)

[5. Método POST en el API Fetch 7](#_Toc189731901)

[Sintaxis básica 7](#_Toc189731902)

[Explicación paso a paso 7](#_Toc189731903)

[Ejemplo con autenticación (Token) 7](#_Toc189731904)

[Manejo de errores en una petición POST 8](#_Toc189731905)

### Introducción

El **API Fetch** es una interfaz moderna y flexible de JavaScript que permite realizar peticiones HTTP de manera sencilla y eficiente. Fue introducido como una alternativa al antiguo XMLHttpRequest, ofreciendo una sintaxis más limpia y basada en **promesas**, lo que facilita el manejo de respuestas asíncronas.

Con Fetch, es posible realizar peticiones a servidores web para obtener o enviar datos en distintos formatos, como **JSON, texto o archivos binarios**. Su uso es común en aplicaciones web que necesitan comunicarse con APIs externas para obtener información en tiempo real.

### Características principales de Fetch:

* **Basado en promesas**: evita el uso de callbacks anidados, mejorando la legibilidad del código.
* **Soporta solicitudes HTTP estándar** (GET, POST, PUT, DELETE, etc.).
* **Permite configurar cabeceras y enviar datos en el cuerpo de la petición**.
* **Funciona en la mayoría de los navegadores modernos**.

**Ejemplo básico de uso:**

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => response.json()) // Convierte la respuesta a JSON

.then(data => console.log(data)) // Muestra los datos en la consola

.catch(error => console.error('Error:', error)); // Manejo de errores

En este ejemplo, fetch() realiza una petición GET a una API pública, y cuando recibe la respuesta, la convierte a JSON antes de imprimirla en la consola.

Fetch también permite manejar errores y trabajar con peticiones más avanzadas, como autenticación, envío de datos y manejo de respuestas en distintos formatos.

### El objeto Response en el API Fetch

Cuando se realiza una petición con fetch(), este devuelve una **promesa** que se resuelve en un objeto de tipo **Response**. Este objeto representa la respuesta HTTP del servidor e incluye información sobre el estado de la respuesta, las cabeceras y los datos devueltos.

**Propiedades del objeto Response**

Algunas propiedades importantes del objeto Response son:

1. **status** – Devuelve el código de estado HTTP de la respuesta.

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => console.log(response.status)); // 200 (OK)

1. **ok** – Retorna true si la respuesta tiene un estado exitoso (200-299), de lo contrario, false.

fetch('https://jsonplaceholder.typicode.com/404')

.then(response => console.log(response.ok)); // false (porque la URL no existe)

1. **statusText** – Texto descriptivo del estado de la respuesta.

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => console.log(response.statusText)); // "OK"

1. **headers** – Permite acceder a las cabeceras de la respuesta.

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => console.log(response.headers.get('Content-Type'))); // "application/json; charset=utf-8"

**Métodos del objeto Response**

El objeto Response tiene varios métodos para extraer los datos en distintos formatos:

1. **response.json()** – Convierte la respuesta en un objeto JavaScript.

fetch('https://jsonplaceholder.typicode.com/posts/1')

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

.then(data => console.log(data)); // { userId: 1, id: 1, title: "...", body: "..." }

1. **response.text()** – Devuelve la respuesta como una cadena de texto.

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => response.text())

.then(data => console.log(data)); // Imprime la respuesta en formato de texto

1. **response.blob()** – Obtiene los datos en formato binario (útil para imágenes o archivos).

fetch('https://via.placeholder.com/150')

.then(response => response.blob())

.then(blob => console.log(blob)); // Blob {size: 1234, type: "image/png"}

1. **response.formData()** – Extrae los datos de un formulario enviado.

fetch('/submit-form', { method: 'POST' })

.then(response => response.formData())

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

1. **response.arrayBuffer()** – Devuelve los datos en un ArrayBuffer, útil para manejar datos binarios de manera más eficiente.

fetch('https://via.placeholder.com/150')

.then(response => response.arrayBuffer())

.then(buffer => console.log(buffer)); // ArrayBuffer { byteLength: 1234 }

**Manejo de errores con Response**

Si una petición falla (por ejemplo, una URL incorrecta), fetch() no lanza un error automáticamente, sino que se debe comprobar response.ok antes de procesar los datos:

fetch('https://jsonplaceholder.typicode.com/posts/123456') // No existe

.then(response => {

if (!response.ok) {

throw new Error(`Error ${response.status}: ${response.statusText}`);

}

return response.json();

})

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

.catch(error => console.error('Error en la petición:', error));

**Conclusión**

El objeto Response es fundamental en el uso del API Fetch, ya que permite acceder a la respuesta HTTP del servidor, verificar su estado y extraer los datos en distintos formatos. Su flexibilidad lo hace ideal para interactuar con APIs en aplicaciones web modernas.

### Métodos de la petición

### Método GET en el API Fetch

El método **GET** es el tipo de petición HTTP más común y se usa para **solicitar** datos de un servidor sin modificarlo. En el API Fetch, una petición GET se realiza de forma predeterminada al llamar a fetch(url), sin necesidad de especificar el método.

### ****Sintaxis básica****

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => response.json()) // Convertir la respuesta a JSON

.then(data => console.log(data)) // Mostrar los datos en la consola

.catch(error => console.error('Error:', error));

**Explicación:**

1. fetch(url): Realiza la petición GET a la URL proporcionada.
2. .then(response => response.json()): Convierte la respuesta en formato JSON.
3. .then(data => console.log(data)): Muestra los datos en la consola.
4. .catch(error => console.error('Error:', error)): Captura posibles errores de red.

### ****Ejemplo con parámetros en la URL****

Si necesitas enviar datos en una petición GET, debes agregarlos en la **query string** (parte después del ? en la URL):

const userId = 1;

fetch(`https://jsonplaceholder.typicode.com/posts?userId=${userId}`)

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

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

.catch(error => console.error('Error:', error));

Aquí se filtran los posts por userId=1.

### ****Ejemplo con cabeceras personalizadas****

A veces, se necesitan cabeceras adicionales, como tokens de autenticación:

fetch('https://api.ejemplo.com/datos', {

method: 'GET',

headers: {

'Authorization': 'Bearer token\_aqui',

'Accept': 'application/json'

}

})

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

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

.catch(error => console.error('Error:', error));

**Cabeceras importantes:**

* **Authorization**: Para autenticar peticiones con un token.
* **Accept**: Indica el tipo de respuesta que esperamos (ej. JSON).

### ****Manejo de errores en una petición GET****

El método fetch() solo lanza un error si hay un problema de red. Para detectar errores en la respuesta (ej. 404 Not Found o 500 Internal Server Error), hay que verificar response.ok:

fetch('https://jsonplaceholder.typicode.com/posts/9999') // Un post que no existe

.then(response => {

if (!response.ok) {

throw new Error(`Error ${response.status}: ${response.statusText}`);

}

return response.json();

})

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

.catch(error => console.error('Error en la petición:', error));

Si el recurso no existe (404), se lanza un error personalizado.

### ****Conclusión****

El método GET con fetch() es la forma más sencilla de obtener datos de un servidor. Sin embargo, es importante manejar correctamente los errores y, si es necesario, incluir parámetros en la URL y cabeceras personalizadas.

### Método POST en el API Fetch

El método **POST** se usa en HTTP para **enviar datos** a un servidor, generalmente para crear nuevos recursos. A diferencia de **GET**, que solo solicita información, **POST** envía datos en el cuerpo de la petición.

En **Fetch**, se debe especificar el método POST y el contenido en el cuerpo de la petición (body), generalmente en formato **JSON**.

### ****Sintaxis básica****

fetch('https://jsonplaceholder.typicode.com/posts', {

method: 'POST', // Especificamos el método

headers: {

'Content-Type': 'application/json' // Indicamos que enviamos JSON

},

body: JSON.stringify({ // Convertimos el objeto a JSON

title: 'Nuevo post',

body: 'Este es el contenido del post',

userId: 1

})

})

.then(response => response.json()) // Convertimos la respuesta a JSON

.then(data => console.log('Post creado:', data))

.catch(error => console.error('Error:', error));

### ****Explicación paso a paso****

1. **fetch(url, opciones)**: Se llama a fetch(), especificando:
   * method: 'POST' → Define que es una petición POST.
   * headers → Se indica que los datos se enviarán en formato JSON.
   * body → Contiene los datos a enviar en formato JSON (convertidos con JSON.stringify()).
2. **.then(response => response.json())** → Convierte la respuesta en un objeto JavaScript.
3. **.then(data => console.log(data))** → Muestra la respuesta del servidor.
4. **.catch(error => console.error(error))** → Manejo de errores.

### ****Ejemplo con autenticación (Token)****

Si el servidor requiere autenticación con un token **Bearer**, se debe incluir en los **headers**:

fetch('https://api.ejemplo.com/crear-usuario', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': 'Bearer mi\_token\_aqui'

},

body: JSON.stringify({

nombre: 'Juan',

email: 'juan@example.com',

password: '123456'

})

})

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

.then(data => console.log('Usuario creado:', data))

.catch(error => console.error('Error:', error));

**Importante**:

* La cabecera **Authorization** permite autenticar la petición.
* Content-Type: application/json indica que se envían datos en JSON.

### ****Manejo de errores en una petición POST****

Como fetch() no lanza errores para respuestas con código HTTP 400 o 500, hay que comprobar **response.ok**:

fetch('https://jsonplaceholder.typicode.com/posts', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ title: '', body: '', userId: 1 }) // Datos inválidos

})

.then(response => {

if (!response.ok) {

throw new Error(`Error ${response.status}: ${response.statusText}`);

}

return response.json();

})

.then(data => console.log('Respuesta:', data))

.catch(error => console.error('Error en la petición:', error));

Si el servidor responde con **400 Bad Request**, el error se captura y se muestra en consola.