18 DE ABRIL DE 2022 / #JAVASCRIPT

Tutorial de Fetch API en JavaScript con ejemplos de JS Fetch, Post y Header

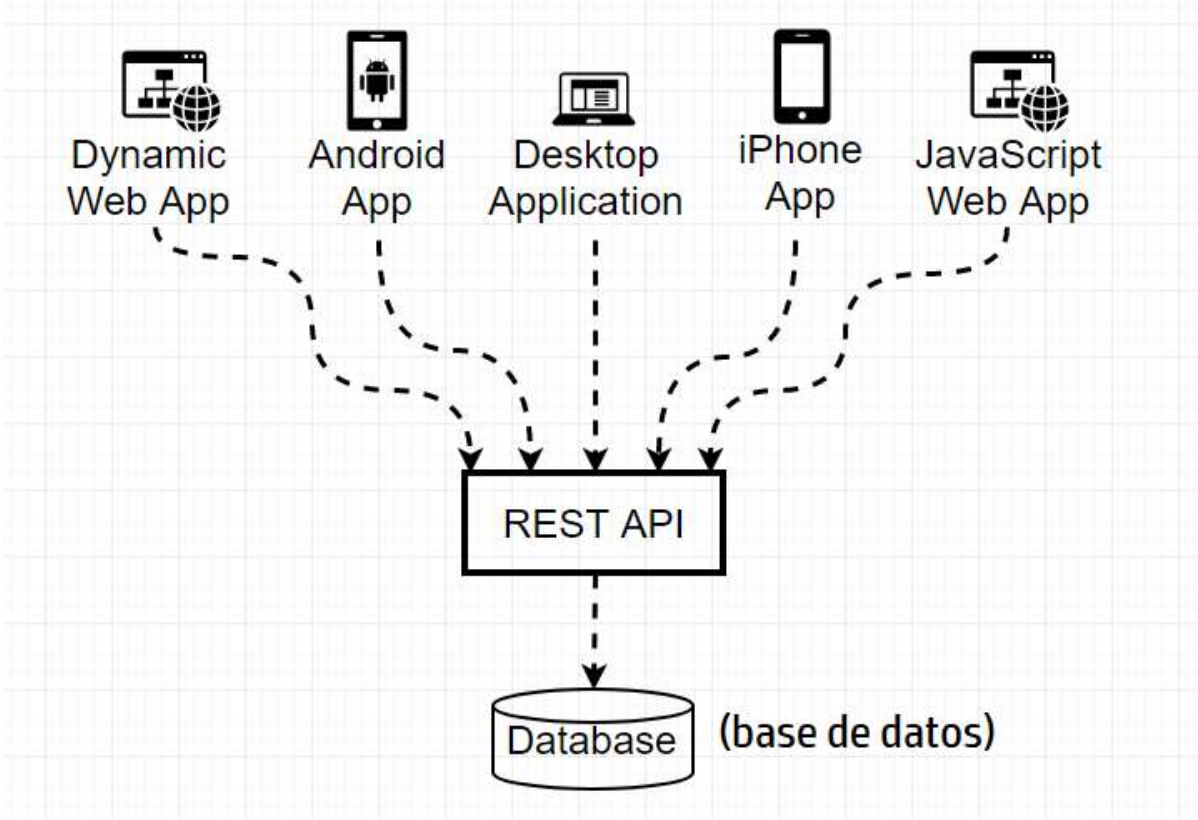
Gemma Fuster

Artículo original escrito por: Manish Shivanandhan
Artículo original: JavaScript Fetch API Tutorial with JS Fetch Post and Header Examples
Traducido y adaptado por: Gemma Fuster

Si estás escribiendo una aplicación web, es probable que tengas que trabajar con datos externos. Puede ser tu propia base de datos, una API de terceros, etc.

Cuando AJAX apareció por primera vez en 1999, nos mostró una forma mejor de construir aplicaciones web. AJAX fue un hito en el desarrollo web y es el concepto central detrás de muchas tecnologías modernas como React.

Antes de AJAX, tenías que volver a renderizar una página web completa, incluso para actualizaciones menores. Pero AJAX nos dio una forma de recuperar el contenido del backend y de actualizar los elementos de la interfaz de usuario seleccionados. Esto ayudó a los desarrolladores a mejorar la experiencia del usuario y a construir plataformas web más grandes y complicadas.



Ahora estamos en la era de las RESTful APIs. En pocas palabras, una REST API te permite enviar y extraer datos de un almacén de datos. Esta podría ser tu base de datos o el servidor de un tercero como Twitter API.

Hay varios tipos de REST APIs. Veamos los que usarás en la mayoría de los casos.

- **GET**—Obtener datos de la API. Por ejemplo, obtener un usuario de Twitter en función de su nombre de usuario.
- **POST**—Empujar los datos a la API. Por ejemplo, crear un nuevo registro de usuario con nombre, edad y dirección de correo electrónico.
- **PUT**—Actualizar un registro existente con nuevos datos. Por ejemplo, actualizar la dirección de correo electrónico de un usuario.
- **DELETE**—Eliminar un registro. Por ejemplo, eliminar un usuario de la base de datos.

Toda REST API tiene tres elementos. La solicitud (request), la respuesta

Request—Estos son los datos que envías a la API, como una identificación de pedido (id) para obtener los detalles del pedido.

```
Manishs-MacBook-Pro:~ manish$ curl -X GET -v https://api.github.com/users/manishmshiva
```

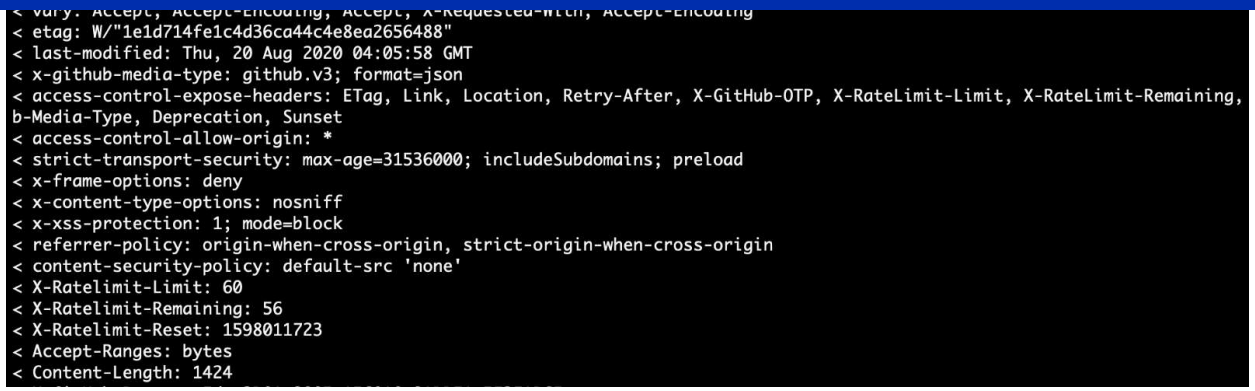
Sample Request

Response—Los datos que obtengas del servidor después de una solicitud exitosa o fallida.


```
{
  "login": "manishmshiva",
  "id": 46276883,
  "node_id": "MDQ6VXNlcjQ2Mjc2ODgz",
  "avatar_url": "https://avatars1.githubusercontent.com/u/46276883?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/manishmshiva",
  "html_url": "https://github.com/manishmshiva",
  "followers_url": "https://api.github.com/users/manishmshiva/followers",
  "following_url": "https://api.github.com/users/manishmshiva/following{/other_user}",
  "gists_url": "https://api.github.com/users/manishmshiva/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/manishmshiva/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/manishmshiva/subscriptions",
  "organizations_url": "https://api.github.com/users/manishmshiva/orgs",
  "repos_url": "https://api.github.com/users/manishmshiva/repos",
  "events_url": "https://api.github.com/users/manishmshiva/events{/privacy}",
  "received_events_url": "https://api.github.com/users/manishmshiva/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Manish M. Shivanandhan",
  "company": null,
  "blog": "http://www.manishmshiva.com/",
  "location": null,
  "email": null,
  "hireable": true,
  "bio": "Cybersecurity | Machine Learning | DevOps",
  "twitter_username": null,
  "public_repos": 3,
  "public_gists": 0,
  "followers": 1,
  "following": 4,
  "created_at": "2018-12-31T08:05:40Z",
  "updated_at": "2020-08-20T04:05:58Z"
}
```


Sample Response

Headers—Metadatos adicionales que se mandan a la API para ayudar al servidor a comprender qué tipo de solicitud se está mandando, por ejemplo, “content-type” (tipo de contenido).



Sample Headers

La gran ventaja de usar una REST API es que puedes crear una sola capa de API para que la usen varias aplicaciones.

Si tienes una base de datos que deseas administrar mediante una aplicación web, móvil y de escritorio, todo lo que necesitas es una única capa de REST API.

Ahora que sabes cómo funcionan las REST APIs, veamos cómo podemos consumirlas.

XMLHttpRequest

Antes de que JSON se convirtiera en lo popular que es hoy, el formato principal del intercambio de datos fue XML. XMLHttpRequest() es una función de JavaScript que hace posible obtener datos de las API que devuelven datos en XML.

XMLHttpRequest nos da la opción de obtener datos XML desde el backend sin recargar toda la página.

Esta función ha crecido desde sus días iniciales cuando era solamente XML. Ahora es compatible con otros formatos de datos como JSON y texto sin formato.

Escribamos una simple llamada XMLHttpRequest a la API de GitHub para

```
// funcion para cuando la llamada es exitosa
function exito() {
    var datos = JSON.parse(this.responseText); //convertir a JSON
    console.log(datos);
}

// funcion para la llamada fallida
function error(err) {
    console.log('Solicitud fallida', err); //los detalles en el objecto "err"
}

var xhr = new XMLHttpRequest(); //invocar nueva instancia de XMLHttpRequest
xhr.onload = exito; // llamar a la funcion exito si exitosa
xhr.onerror = error;  // llamar a la funcion error si fallida
xhr.open('GET', 'https://api.github.com/users/manishmshiva'); // Abrir solicit
xhr.send(); // mandar la solicitud al vervidor.
```

El código anterior enviará una solicitud GET a https://api.github.com/users/manishmshiva para obtener mi información de GitHub en JSON. Si la respuesta es exitosa, imprimirá el siguiente JSON en la consola:

Si la solicitud falla, imprimirá este mensaje de error en la consola:

Fetch API

La Fetch API es una versión más simple y fácil de usar para consumir recursos de forma asíncrona que una XMLHttpRequest . Fetch te permite trabajar con REST APIs con opciones adicionales, como almacenar datos en caché, leer respuestas de transmisión y más.

La diferencia principal es que Fetch funciona con promesas, no con devoluciones de llamada. Los desarrolladores de JavaScript se han alejado de las devoluciones de llamadas después de la introducción de las promesas.

Para una aplicación compleja, es posible que adquieras fácilmente el hábito de escribir devoluciones de llamadas (callbacks) que conduzcan a un infierno de llamadas.

Con las promesas, es fácil escribir y manejar solicitudes asincrónicas. Si no estás familiarizado con las promesas, puedes aprender como funcionan aqui.

Así es como se quedaría la función que escribimos anteriormente usando fetch() en lugar de XMLHttpRequest:

```
// Solicitud GET (Request).
```

```
    .catch(err => console.log('Solicitud fallida', err)); // Capturar errores
```

El primer parámetro de la función Fetch siempre debe ser la URL. Fetch entonces toma un segundo objeto JSON con opciones como method, headers, request body, etc.

Hay una diferencia importante entre el objeto de respuesta en XMLHttpRequest y Fetch.

XMLHttpRequest devuelve los datos como respuesta, mientras que el objeto de respuesta de Fetch contiene información sobre el objeto de respuesta en sí mismo. Esto incluye headers, status code, etc. Llamamos a la función “res.json()” para obtener los datos que necesitamos del objeto de respuesta.

Otra diferencia importante es que Fetch API no generará un error si la solicitud devuelve un código de estado 400 o 500. Todavía se marcará como una respuesta exitosa y se pasará a la función `.then`.

Fetch solo arroja un error si la solicitud en sí se interrumpe. Para manejar respuestas 400 y 500, puedes escribir una lógica personalizada usando 'response.status'. La propiedad 'status' te dará el código de estado de la respuesta devuelta.

Estupendo. Ahora que sabes cómo funciona la Fetch API, veamos un par de ejemplos más, sobre como pasar datos y trabajar con headers.

Trabajando con Headers

Puedes pasar encabezados con la propiedad “headers”. Pasar un objeto JSON a la propiedad "headers" debería funcionar en la mayoría de los casos.

```
  headers: {"Content-type": "application/json;charset=UTF-8"}
})
.then(response => response.json())
.then(json => console.log(json));
.catch(err => console.log(err));
```

Pasando datos a una solicitud POST

Para una solicitud POST, puedes usar la propiedad “body” para pasar una cadena JSON como input. Date cuenta de que el "body" debe ser una cadena JSON, pero los encabezados (headers) deben ser un objecto JSON.

```
// datos mandados con la solicutud POST
let _datos = {
  titulo: "foo",
  principal: "bar",
  Id:1
}

fetch('https://jsonplaceholder.typicode.com/posts', {
  method: "POST",
  body: JSON.stringify(_datos),
  headers: {"Content-type": "application/json; charset=UTF-8"}
})
.then(response => response.json())
.then(json => console.log(json));
.catch(err => console.log(err));
```

La Fetch API todavía está en desarrollo. Podemos esperar mejoras en un futuro próximo.

Sin embargo, la mayoría de los navegadores admiten el uso de Fetch en sus aplicaciones. La tabla a continuación debería ayudarte a determinar qué navegadores lo admiten en la web y las aplicaciones móviles.

	Chrom	Edge	Firefox	Interne	Opera	Safari	Androi	Chrom	Firefox	Opera	Safari	Samsu
fetch	42	14	39	No	29	10.1	42	42	39	29	10.3	4.0
Support for blob: and data:	48	79	?	No	?	?	43	48	?	?	?	5.0
referrerPolicy	52	79	52	No	39	11.1	52	52	52	41	No	6.0
signal	66	16	57	No	53	11.1	66	66	57	47	11.3	9.0
Streaming response body	43	14	Yes	No	29	10.1	43	43	No	No	10.3	4.0

Espero que este artículo te haya ayudado a comprender cómo trabajar con la API Fetch. Asegúrate de probar Fetch en tu próxima aplicación web.

Gemma Fuster

Programmer. Had to be off work for a while (thanks pandemic) but getting ready to get back into it very soon. I also enjoy translating articules here and there.

Si leíste hasta aquí, haz un tweet al autor para mostrarle que te importa su trabajo. Realiza un Tweet de agradecimiento

Aprende a codificar de forma gratuita. El plan de estudios de código abierto de freeCodeCamp ha ayudado a más de 40,000 personas a obtener trabajos como desarrolladores. Empieza

Los pros y los contras de
colocación e interconexi

Equinix

freeCodeCamp es una organización sin fines de lucro exenta de impuestos 501(c)(3) respaldada por donantes (Número de identificación fiscal federal de los Estados Unidos: 82-0779546)

Nuestra misión: ayudar a las personas a aprender a programar de forma gratuita. Logramos esto mediante la creación de miles de videos, artículos y lecciones de programación interactivas, todos disponibles gratuitamente para el público. También tenemos miles de grupos de estudio de FreeCodeCamp en todo el mundo.

Las donaciones a freeCodeCamp se destinan a nuestras iniciativas educativas y ayudan a pagar los servidores, los servicios y el personal.

Puedes hacer una donación deducible de impuestos aquí.

Guías de tendencias

Git Clone

Métodos Agile

Python Main

Callback

Debounce

URL Encode

Blink HTML

Python Tupla

JavaScript Push

Java List

UX

Proceso de Diseño

Números Primos

Diseño de Producto

Digital Design

Juegos de Código

SVM

JavaScript forEach

Google BERT

Create Table SQL

¿Qué es un Archivo PDF?

What Is Python?

Ejemplos de RSync

Random Forest

Nuestra organización sin fines de lucro