

30-10-2023

# Programación Asíncrona

## Investigación sobre Fetch y Promesas



Jorge Eduardo Escobar Bugarini - 21550317  
TECNOLÓGICO NACIONAL DE MÉXICO  
INSTITUTO TECNOLÓGICO DE CHIHUAHUA II  
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN  
INGENIERÍA EN SISTEMAS COMPUTACIONALES  
PW-10-ISC  
PROGRAMACIÓN WEB - 10-11  
EVALUACIÓN - 3  
DOCENTE: LUIS ARMANDO ACOSTA RAMÍREZ

## Contenido

<b>¿Qué es Fetch y para qué se utiliza? .....</b>	<b>1</b>
<b>Sintaxis básica de Fetch y ejemplos .....</b>	<b>1</b>
Comprobar una solicitud de fetch .....	3
<b>Proporcionando objetos Request .....</b>	<b>4</b>
Utilizando credenciales include .....	4
<b>Enviando datos JSON y otros archivos .....</b>	<b>5</b>

## ¿Qué es Fetch y para qué se utiliza?

Fetch es una API incorporada en JavaScript que permite realizar solicitudes de red de manera asíncrona. Fue introducida en el estándar ES6 (ECMAScript) y ha ganado una gran popularidad desde entonces debido a su simplicidad y flexibilidad. Fetch reemplazó en gran medida a la antigua API XMLHttpRequest, ofreciendo una interfaz más limpia y fácil de usar para interactuar con servidores web y recuperar recursos. La función Fetch se utiliza para realizar solicitudes HTTP y devuelve una promesa que se resuelve en la respuesta de la solicitud.

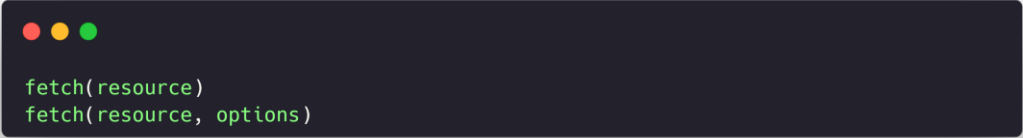
## Sintaxis básica de Fetch y ejemplos

La API Fetch puede recibir dos parámetros, el recurso (el lugar de donde buscará la información) y sus

opciones (estas son opcionales, pero de gran ayuda); las opciones que esta puede recibir son:

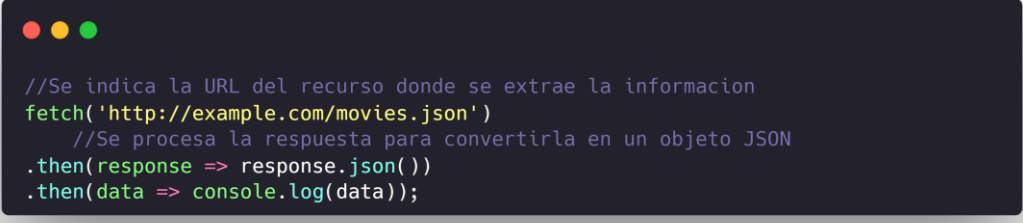
- **method:** el tipo de solicitud que se quiere hacer, este puede ser GET, POST, PUT, DELETE, etc.
- **mode:** Se utiliza para determinar si las solicitudes de origen cruzado conducen a respuestas
- **válidas,** y qué propiedades de la respuesta son legibles (same-origin, no-cors, cors, navigate & websocket).
- **cache:** Una cadena que indica cómo interactuará la petición con la caché HTTP del navegador (default, no-store, reload, no-cache, force-cache & only-if-cached).
- **credentials:** Controla lo que hacen los navegadores con las credenciales como: cookies, autenticación HTTP y certificados de cliente TLS (omit, same-origin & include).
- **headers:** Cualquier encabezado que desee agregar a su solicitud, contenido dentro de un objeto Headers o un objeto literal con valores String.

- redirect: Indica cómo gestionar una respuesta de redirección (follow, error, manual).
- referrerPolicy: Especifica la política de referencia que se utilizará para la solicitud (no-referrer, no-referrer-when-downgrade, same-origin, origin, strict-origin, origin-when-cross-origin, strict origin-when-cross-origin & unsafe-url)



```
fetch(resource)
fetch(resource, options)
```

En el siguiente ejemplo estamos buscando información de la página “example.com” la cual nos arroja una respuesta HTTP, por ello utilizamos el método “json()” para poder obtener el objeto e imprimirlo en consola



```
//Se indica la URL del recurso donde se extrae la informacion
fetch('http://example.com/movies.json')
//Se procesa la respuesta para convertirla en un objeto JSON
.then(response => response.json())
.then(data => console.log(data));
```

En el siguiente ejemplo podemos ver que además de utilizar el recurso de donde se buscará la información también se le mandan sus opciones como que el método será tipo POST o que el método de redirección será “follow”, estas tienen una variedad de configuraciones que hacen que Fetch sea muy útil

```

async function postData(url = '', data = {}){
  const response = await fetch(url,{
    method:'POST',
    mode:'cors',
    cache:'no-cache',
    credentials:'same-origin',
    headers:{'Content-type':'application/json'},
    redirect:'follow',
    referrerPolicy:'no-referrer',
    body: JSON.stringify(data)
  });
  return response.json();
}

postData('https://example.com/answer', { answer: 42})
  .then(data => console.log(data));

```

### Comprobar una solicitud de fetch

Una petición Fetch() puede ser rechazada por un error de red, por ello es necesario corroborar que la petición resuelva correctamente la promesa, además se comprueba la propiedad "Response.ok" esta tiene el valor true y se utiliza de la siguiente manera

```

fetch("flores.jpg")
  .then(function (response) {
    if (response.ok) {
      response.blob().then(function (miBlob) {
        var objectURL = URL.createObjectURL(miBlob);
        miImagen.src = objectURL;
      });
    } else {
      console.log("Respuesta de red OK pero respuesta HTTP no OK");
    }
  })
  .catch(function (error) {
    console.log("Hubo un problema con la petición Fetch:" + error.message);
  });

```

## Proporcionando objetos Request

También podemos crear un objeto de tipo Request para especificar el recurso y las opciones que deseamos que tenga la Fetch() API, en el siguiente ejemplo podemos ver que se declara un objeto de tipo Headers, una variable myInit y un objeto de tipo Request con los cuales declaramos la configuración del Fetch()

```
var myHeaders = new Headers();

var myInit = {
  method: "GET",
  headers: myHeaders,
  mode: "cors",
  cache: "default",
};

var myRequest = new Request("flowers.jpg", myInit);

fetch(myRequest)
  .then(function (response) {
    return response.blob();
  })
  .then(function (myBlob) {
    var objectURL = URL.createObjectURL(myBlob);
    myImage.src = objectURL;
  });
```

### Utilizando credenciales include

La opción “credentials: include” hace que se incluyan las credenciales en la solicitud, independientemente del origen de la solicitud. Se utiliza cuando necesitas enviar cookies o encabezados de autenticación a un dominio diferente al del documento actual

```
fetch("https://example.com", {  
  credentials: "include",  
});
```

## Enviando datos JSON y otros archivos

A su vez también se pueden utilizar JSON y archivos en las Fetch() para adaptar la API a nuestra conveniencia, esto puede ser de la siguiente manera:

```
var url = "https://example.com/profile";  
var data = { username: "example" };  
  
fetch(url, {  
  method: "POST", // or 'PUT'  
  body: JSON.stringify(data), // data can be `string` or {object}!  
  headers: {  
    "Content-Type": "application/json",  
  },  
})  
  .then((res) => res.json())  
  .catch((error) => console.error("Error:", error))  
  .then((response) => console.log("Success:", response));
```



```
var formData = new FormData();
var fileField = document.querySelector("input[type='file']");

formData.append("username", "abc123");
formData.append("avatar", fileField.files[0]);

fetch("https://example.com/profile/avatar", {
  method: "PUT",
  body: formData,
})
.then((response) => response.json())
.catch((error) => console.error("Error:", error))
.then((response) => console.log("Success:", response));
```