UD6

Utilización de mecanismos de comunicación asíncrona: Fetch

1. Fetch: un nuevo mecanismo de comunicación asíncrona 1

2. Funcionamiento básico 2

2.1 Opciones de fetch(). 3

2.2. Parámetros del headers (cabecera) de nuestra petición. 4

2.3. Métodos consumidores de respuesta. 4

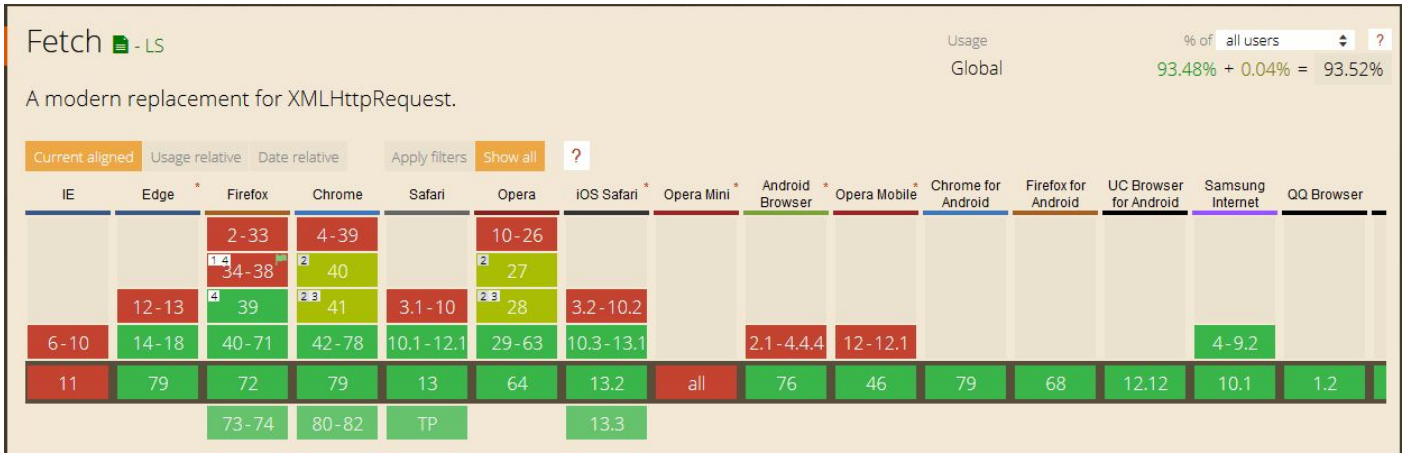
3. Enlaces 4

1. Fetch: un nuevo mecanismo de comunicación asíncrona

El **API fetch** es un nuevo estándar que viene a dar una alternativa para interactuar por HTTP, con un diseño moderno, **basado en promesas**, con mayor flexibilidad y capacidad de control a la hora de realizar llamadas al servidor.

También está disponible en **Node**, por lo que podemos **utilizarlo de forma isomórfica**, es decir, tanto en cliente como en servidor.

Está ampliamente soportado por la gran mayoría de los navegadores actuales:



2. Funcionamiento básico

Una de las características más importantes del API fetch es que **utiliza promesas**, es decir, devuelve un objeto con dos métodos, uno then() y otro catch() a los que pasaremos una función que será invocada cuando se obtenga la respuesta o se produzca un error.

Importante: Hay que aclarar un punto con los errores: si se devuelve un código HTTP correspondiente a un error, es decir una respuesta errónea del servidor porque, por ejemplo, se han pasado mal los parámetros de la petición, **no se ejecutará el catch()**, ya que se ha obtenido una respuesta válida, por lo que se ejecutará el then(). **Sólo si hay un error de red o de otro tipo se ejecutará el catch().**

Otro aspecto importante que hay que comprender es que **para obtener el body o cuerpo del mensaje devuelto por el servidor deberemos obtener una segunda promesa** por medio de los métodos del objeto Response. **Por ello será muy habitual ver dos promesas encadenadas**, una para el fetch() y otra con el retorno del método que utilicemos para obtener el body.

En los ejemplos vamos a utilizar las opciones que nos ofrece https://httpbin.org/ . Esta es una API gratuita que se puede utilizar para testear todo tipo de casuísticas: hacer peticiones get, post, etc, que devuelvan un solo dato, json, imágenes, etc. Échale un vistazo a la API.

Vamos a verlo algunos ejemplos:

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<meta http-equiv="X-UA-Compatible" content="ie=edge">
	<title>Documento</title>
</head>
<body>
	<script>
	fetch('https://httpbin.org/ip')
	.then(function(response) {
		return response.text();
	})
	.then(function(data) {
		console.log('data = ', data);
	})
	.catch(function(err) {
		console.error(err);
	});
	</script>
</body>
</html>
```

¿Qué ha "pachao" aquí?

1º. Hemos llamado a fetch() con la URL a la que queremos acceder como parámetro y esta llamada nos devuelve una promesa

2º. El método then() de esa promesa nos entrega un objeto response.

3º. Del objeto response llamamos al método text() para obtener el cuerpo retornado en forma de texto y este método nos devuelve otra promesa que se resolverá cuando se haya obtenido el contenido.

4º. El método then() de esa promesa recibe el cuerpo devuelto por el servidor en formato de texto.

5º. hemos incluido un catch() por si se produce algún error.

2.1. Opciones de fetch()

Quizás no nos hayamos dado cuenta, pero en el ejemplo anteriores no hemos indicado que método teníamos que utilizar, simplemente hemos pasado la URL y se considera que queremos utilizar el método **GET que es el valor por defecto**. La forma de configurar esta llamada es utilizar el segundo parámetro de fetch(), donde pasaremos un objeto con las opciones que deseemos. Veámoslo con un ejemplo:

```
fetch('https://httpbin.org/post',
   {
      method: 'POST',
      headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
      },
      body: 'a=1&b=2'
   })
   .then(function(response) {
         console.log('response =', response);
         return response.json();
         console.log('data = ', data);
   })
    .then(function(data) {
         console.log('data = ', data);
   })
     .catch(function(err) {
         console.error(err);
   });
```

Fíjate especialmente en el segundo parámetro:

```
{
      method: 'POST',       // Especificamos el método de la petición.
      headers: {            // Podemos añadir un header con una serie de atributos.
            'Content-Type': 'application/x-www-form-urlencoded'
      },
```

```
      body: 'a=1&b=2'       //Añadimos los parámetros de la petición POST
}
```

2.2. Parámetros del headers (cabecera) de nuestra petición. INFORMATIVO

Aquí detallamos lista de parámetros que pueden estar incluidos en el Header de nuestra petición:

Accept-Charset, Accept-Encoding	Keep-Alive
Access-Control-Request-Headers	Origin
Access-Control-Request-Method	Referer
Connection	TE
Content-Length	Trailer
Cookie, Cookie2	Transfer-Encoding
Date	Upgrade
DNT	Via
Expect	Proxy-*
Host	Sec-*

2.3. Métodos consumidores de respuesta.

A continuación detallaremos una serie de métodos que se utilizamos para interpretar/usar la respuesta de nuestra petición. Estos métodos están basados en "**promesas**". Los más utilizados son los siguientes:

- **response.text()** – Devuelve la respuesta como texto.
- **response.json()** – Parsea la respuesta como JSON object.
- **response.formData()** – Devuelve la respuestas como objeto de tipo FormData.
- **response.blob()** – Devuelve la respuesta como Blob (binary data with type),
- **response.arrayBuffer()** – Devuelve la respuesta como ArrayBuffer (low-level binary data).

Todos son métodos del objeto **Response.** Son muchos más métodos y propiedades que podemos obtener de este objeto. En el siguiente enlace puedes ver una referencia completa del mismo:

- [Objeto Response]

2.4. Deshabilitar validación CORS usando Fetch:

Debemos de usar el parámetro “mode” donde seteamos como valor “no-cors”.

```
mode: 'cors',     // no-cors,     *cors,    same-origin
```

Enlaces

Buen y simple manual: https://scotch.io/tutorials/how-to-use-the-javascript-fetch-api-to-get-data