Fetch

JavaScript puede enviar peticiones de red al servidor y cargar nueva información siempre que se necesite.

Por ejemplo, podemos utilizar una petición de red para:

- Crear una orden,
- Cargar información de usuario,
- Recibir las últimas actualizaciones desde un servidor,
- ...etc.

...Y todo esto sin la necesidad de refrescar la página.

Se utiliza el término global "AJAX" (abreviado **A**synchronous **J**avaScript **A**nd **X**ML, en español: "JavaScript y XML Asincrónico") para referirse a las peticiones de red originadas desde JavaScript. Sin embargo, no estamos necesariamente condicionados a utilizar XML dado que el término es antiguo y es por esto que el acrónimo XML se encuentra aquí. Probablemente lo hayáis visto anteriormente.

Existen múltiples maneras de enviar peticiones de red y obtener información de un servidor.

Comenzaremos con el el método `fetch()` que es moderno y versátil. Este método no es soportado por navegadores antiguos (sin embargo se puede incluir un polyfill), pero es perfectamente soportado por los navegadores actuales y modernos.

La sintaxis básica es la siguiente:

```
1  let promise = fetch(url, [options])
```

- **`url`** – representa la dirección URL a la que deseamos acceder.
- **`options`** – representa los parámetros opcionales, como puede ser un método o los encabezados de nuestra petición, etc.

Si no especificamos ningún `options` , se ejecutará una simple petición GET, la cual descargará el contenido de lo especificado en el `url` .

El navegador lanzará la petición de inmediato y devolverá una promesa (promise) que luego será utilizada por el código invocado para obtener el resultado.

Por lo general, obtener una respuesta es un proceso de dos pasos.

Primero, la promesa `promise` , devuelta por `fetch` , resuelve la respuesta con un objeto de la clase incorporada Response tan pronto como el servidor responde con los encabezados de la petición.

En este paso, podemos chequear el status HTTP para poder ver si nuestra petición ha sido exitosa o no, y chequear los encabezados, pero aún no disponemos del cuerpo de la misma.

La promesa es rechazada si el `fetch` no ha podido establecer la petición HTTP, por ejemplo, por problemas de red o si el sitio especificado en la petición no existe. Estados HTTP anormales, como el 404 o 500 no generan errores.

Podemos visualizar los estados HTTP en las propiedades de la respuesta:

- **`status`** – código de estado HTTP, por ejemplo: 200.
- **`ok`** – booleana, `true` si el código de estado HTTP es 200 a 299.

Ejemplo:

```
let response = await fetch(url);

if (response.ok) { // si el HTTP-status es 200-299
  // obtener cuerpo de la respuesta (método debajo)
  let json = await response.json();
} else {
  alert("Error-HTTP: " + response.status);
}
```

Segundo, para obtener el cuerpo de la respuesta, necesitamos utilizar un método adicional.

`Response` provee múltiples métodos basados en promesas para acceder al cuerpo de la respuesta en distintos formatos:

- **`response.text()`** – lee y devuelve la respuesta en formato texto,
- **`response.json()`** – convierte la respuesta como un JSON,
- **`response.formData()`** – devuelve la respuesta como un objeto `FormData` (explicado en el siguiente capítulo),
- **`response.blob()`** – devuelve la respuesta como Blob (datos binarios tipados),
- **`response.arrayBuffer()`** – devuelve la respuesta como un objeto ArrayBuffer (representación binaria de datos de bajo nivel),
- Adicionalmente, `response.body` es un objeto ReadableStream, el cual nos permite acceder al cuerpo como si fuera un stream y leerlo por partes. Veremos un ejemplo de esto más adelante.

Por ejemplo, si obtenemos un objeto de tipo JSON con los últimos commits de GitHub:

```
let url = 'https://api.github.com/repos/javascript-tutorial/en.javascript.info
let response = await fetch(url);

let commits = await response.json(); // leer respuesta del cuerpo y devolver c

alert(commits[0].author.login);
```

O también usando promesas, en lugar de `await`:

```
fetch('https://api.github.com/repos/javascript-tutorial/en.javascript.info/com
  .then(response => response.json())
  .then(commits => alert(commits[0].author.login));
```

Para obtener la respuesta como texto, `await response.text()` en lugar de `.json()` :

```
let response = await fetch('https://api.github.com/repos/javascript-tutorial/e

let text = await response.text(); // leer cuerpo de la respuesta como texto

alert(text.slice(0, 80) + '...');
```

Como demostración de una lectura en formato binario, hagamos un fetch y mostremos una imagen del logotipo de "especificación fetch" (ver capítulo Blob para más detalles acerca de las operaciones con `Blob`):

```
let response = await fetch('/article/fetch/logo-fetch.svg');

let blob = await response.blob(); // download as Blob object

// crear tag <img> para imagen
let img = document.createElement('img');
img.style = 'position:fixed;top:10px;left:10px;width:100px';
document.body.append(img);

// mostrar
img.src = URL.createObjectURL(blob);

setTimeout(() => { // ocultar luego de tres segundos
  img.remove();
  URL.revokeObjectURL(img.src);
}, 3000);
```

Importante:

Podemos elegir un solo método de lectura para el cuerpo de la respuesta.

Si ya obtuvimos la respuesta con `response.text()` , entonces `response.json()` no funcionará, dado que el contenido del cuerpo ya ha sido procesado.

```
let text = await response.text(); // cuerpo de respuesta obtenido y procesa
let parsed = await response.json(); // fallo (ya fue procesado)
```

Encabezados de respuesta

Los encabezados de respuesta están disponibles como un objeto de tipo Map dentro del `response.headers` .

No es exactamente un Map, pero posee métodos similares para obtener de manera individual encabezados por nombre o si quisiéramos recorrerlos como un objeto:

```
let response = await fetch('https://api.github.com/repos/javascript-tutorial/e

// obtenemos un encabezado
alert(response.headers.get('Content-Type')); // application/json; charset=utf-

// iteramos todos los encabezados
for (let [key, value] of response.headers) {
  alert(`${key} = ${value}`);
}
```

Encabezados de petición

Para especificar un encabezado en nuestro `fetch` , podemos utilizar la opción `headers` . La misma posee un objeto con los encabezados salientes, como se muestra en el siguiente ejemplo:

```
let response = fetch(protectedUrl, {
  headers: {
  Authentication: 'secret'4
  }
});
```

...Pero existe una lista de encabezados que no pueden ser especificados:

- `Accept-Charset` , `Accept-Encoding`
- `Access-Control-Request-Headers`
- `Access-Control-Request-Method`
- `Connection`
- `Content-Length`
- `Cookie` , `Cookie2`
- `Date`
- `DNT`
- `Expect`
- `Host`
- `Keep-Alive`
- `Origin`
- `Referer`
- `TE`

- `Trailer`
- `Transfer-Encoding`
- `Upgrade`
- `Via`
- `Proxy-*`
- `Sec-*`

Estos encabezados nos aseguran que nuestras peticiones HTTP sean controladas exclusivamente por el navegador, de manera correcta y segura.

Peticiones POST

Para ejecutar una petición `POST`, o cualquier otro método, utilizaremos las opciones de `fetch`:

- **`method`** – método HTTP, por ej: `POST`,
- **`body`** – cuerpo de la respuesta, cualquiera de las siguientes:
  - cadena de texto (ej. JSON-encoded),
  - Objeto `FormData`, para enviar información como `multipart/form-data`,
  - `Blob` / `BufferSource` para enviar información en formato binario,
  - URLSearchParams, para enviar información en cifrado `x-www-form-urlencoded` (no utilizado frecuentemente).

El formato JSON es el más utilizado.

Por ejemplo, el código debajo envía la información `user` como un objeto JSON:

```
let user = {
  nombre: 'Juan',
  apellido: 'Perez'
};

let response = await fetch('/article/fetch/post/user', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  body: JSON.stringify(user)
});

let result = await response.json();
alert(result.message);
```

Tener en cuenta, si la respuesta del `body` es una cadena de texto, entonces el encabezado `Content-Type` será especificado como `text/plain;charset=UTF-8` por defecto.

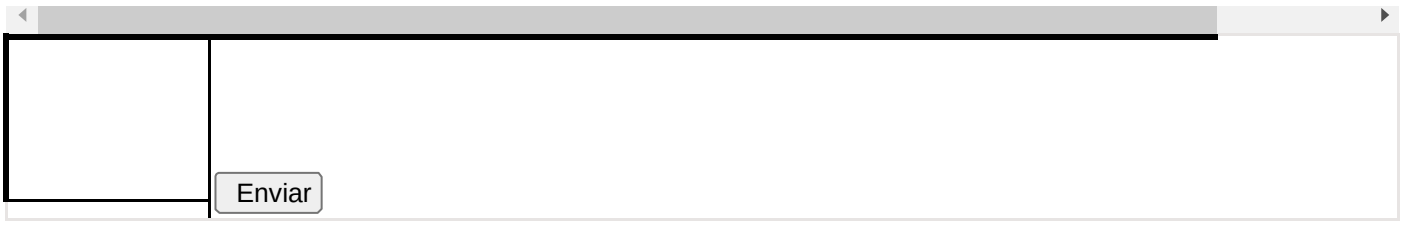
Pero, cómo vamos a enviar un objeto JSON, en su lugar utilizaremos la opción `headers` especificada a `application/json`, que es la opción correcta `Content-Type` para información en formato JSON.

Enviando una imagen

También es posible enviar datos binarios con `fetch`, utilizando los objetos `Blob` o `BufferSource`.

En el siguiente ejemplo, utilizaremos un `<canvas>` donde podremos dibujar utilizando nuestro ratón. Haciendo click en el botón "enviar" enviará la imagen al servidor:

```
<body style="margin:0">
  <canvas id="canvasElem" width="100" height="80" style="border:1px solid"></c

  <input type="button" value="Enviar" onclick="submit()">

  <script>
    canvasElem.onmousemove = function(e) {
      let ctx = canvasElem.getContext('2d');
      ctx.lineTo(e.clientX, e.clientY);
    ctx.stroke();11
    };

    async function submit() {
      let blob = await new Promise(resolve => canvasElem.toBlob(resolve, 'imag
      let response = await fetch('/article/fetch/post/image', {
        method: 'POST',
      body: blob18
      });

      // el servidor responde con una confirmación y el tamaño de nuestra imag
      let result = await response.json();
    alert(result.message);23
    }

  </script>
</body>
```



Una aclaración, aquí no especificamos el `Content-Type` de manera manual, precisamente porque el objeto `Blob` posee un tipo incorporado (en este caso `image/png`, el cual es generado por la función `toBlob`). Para objetos `Blob` ese es el valor por defecto del encabezado `Content-Type`.

Podemos reescribir la función `submit()` sin utilizar `async/await` de la siguiente manera:

```
function submit() {
  canvasElem.toBlob(function(blob) {
    fetch('/article/fetch/post/image', {
      method: 'POST',
```

```
    body: blob
  })
    .then(response => response.json())
    .then(result => alert(JSON.stringify(result, null, 2)))
}, 'image/png');
}
```

Resumen

Una petición fetch típica está formada por dos llamadas `await` :

```
let response = await fetch(url, options); // resuelve con los encabezados de r
let result = await response.json(); // accede al cuerpo de respuesta como json
```

También se puede acceder sin utilizar `await` :

```
fetch(url, options)
  .then(response => response.json())
  .then(result => /* procesa resultado */)
```

Propiedades de respuesta:

- `response.status` – Código HTTP de la respuesta.
- `response.ok` – Devuelve `true` si el código HTTP es 200-299.
- `response.headers` – Objeto simil-Map que contiene los encabezados HTTP.

Métodos para obtener el cuerpo de la respuesta:

- **`response.text()`** – lee y devuelve la respuesta en formato texto,
- **`response.json()`** – convierte la respuesta como un JSON,
- **`response.formData()`** – devuelve la respuesta como un objeto `FormData` (codificación `multipart/form-data` , explicado en el siguiente capítulo),
- **`response.blob()`** – devuelve la respuesta como Blob (datos binarios tipados),
- **`response.arrayBuffer()`** – devuelve la respuesta como un objeto ArrayBuffer (datos binarios de bajo nivel)

Opciones de fetch hasta el momento:

- `method` – método HTTP,
- `headers` – un objeto los encabezados de la petición (no todos los encabezados están permitidos),
- `body` – los datos/información a enviar (cuerpo de la petición) como `string` , `FormData` , `BufferSource` , `Blob` u objeto `UrlSearchParams` .

En los próximos capítulos veremos más sobre opciones y casos de uso para `fetch` .