FETCH API

Antes de fetch: **XMLHttpRequest**

```
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
  request = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
  try {
    request = new ActiveXObject('Msxml2.XMLHTTP');
  }
  catch (e) {
    try {
      request = new ActiveXObject('Microsoft.XMLHTTP');
    }
    catch (e) {}
  }


request.open('GET', 'https://localhost:3000', true);
request.send(null);
```

¿Por qué fetch?

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

- Mucho menos boilerplate
- Funciona con promesas
- No necesita enviar cookies
- No fuerza a utilizar CORS- podemos utilizar una respuesta que no implemente CORS - No se puede utilizar directamente pero se puede redireccionar a otras APIs, o usar los headers
- Cache API

¿Que no se puede hacer aún?

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

- Abortar una petición (Firefox 57, Edge 16)
- Ver el progreso de la petición

Uso básico

fetch(url, options) - Devuelve una promesa

```
fetch('https://keepcoding.com').then((response) => {
    return response.json();
  }.then(data) {    // use data  }
  }).catch(function(err) {    // Error :(  });
```

Opciones

- method - GET, POST, PUT, DELETE, HEAD
- url - URL
- headers - Headers a incluir en la llamada
- referrer - Especificar remitente
- mode - cors, no-cors, same-origin
- credentials - Añadir cookies en la petición? omit, same-origin
- redirect - follow, error, manual
- integrity - valor de integridad para chequearla
- cache - modo de caché (default, reload, no-cache)

¿Qué devuelve response?

- type - basic, cors
- url
- useFinalURL - Booleano si la url es final o no (puede haber encadenadas)
- status - código de estado (por ejemplo: 200, 404, etc.)
- ok - Si la respuesta es un éxito (status entre 200-299)
- statusText - código del status en texto (ex: OK)
- headers - Headers object associados con la respuesta

Además response devuelve

- `json()` - Devuelve una promesa que se resuelve con un JSON de los datos
- `blob()` - Devuelve una promesa que se resuelve con un Blob de los datos (para archivos)
- `text()` - Devuelve una promesa que se resuelve con un USVString (texto).
- `arrayBuffer()` - Devuelve una promesa que se resuelve con un ArrayBuffer
- `error()` - Devuelve una respuesta asociado con un error de red
- `clone()` - Crea un clon de la respuesta
- `redirect()` - Crea una nueva respuesta redireccionando a otra URL
- `formData()` - Devuelve una promesa que se resuelve con un objeto FormData object

Que es CORS (cross-origin resource sharing)

Los servidores no pueden aceptar peticiones de todos los navegadores:

fetch - Delete - all data - from server 

Para ello se establecen algunas políticas de seguridad:

Estas políticas se basan en :

- Cual es el origen de la petición (same-origin, *, ...)
- Qué método se utiliza (GET, POST, DELETE, PATCH)

Que es CORS (cross-origin resource sharing)

Estas políticas se establecen en el servidor:

- Access-Control-Allow-Origin : *, ‘www.keepcoding.com’, ...
- Access-Control-Allow-Methods : ‘POST’, ‘PATCH’
- Access-Control-Allow-Headers
- ...

Filtrarán el acceso exterior desde distintos frontend al backend

Haciendo otras peticiones distintas a GET

```
fetch('https://keepcoding.com', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({ name: ‘Peter’ })
  }).then(...
```