

# JavaScript Basic

### Guía #4: Asincronía, lectura de archivos, peticiones HTTP

Esta guía hace parte del curso mencionado en el título y les permitirá estudiar con más detenimiento los conceptos de clase, además de servir como instrumento de la misma. La guía contará con código base, el desarrollo conceptual de los temas y enlaces externos para el estudio independiente.

## 1. [Fetch API](https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fetch)

La API Fetch proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, como peticiones y respuestas. También provee un método global fetch() que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red. –MDN Mozilla–

Con Fetch API dejamos atrás la necesidad de usar las tediosas [**XMLHTTPRequest**](https://developer.mozilla.org/en/XMLHttpRequest) de manera nativa, es decir, para acceder a un archivo llamado westeros.json alojado en un servidor y ubicado en la ruta http://miservidor/datasets/westeros.js una implementación con XMLHTTPRequest sería así:

var req = new XMLHttpRequest();  
req.open('GET', 'http://127.0.0.1:8887/datasets/westeros.json', false);  
req.send(null);  
if (req.status == 200) {  
 console.log(JSON.parse(req.responseText));  
}

Mientas que la misma implementación con el Fetch API se vería de la siguiente manera:

fetch(`http://127.0.0.1:8887/datasets/westeros.json`,{})  
.then(function(response) {  
 return response.json();  
})  
.then(function(json) {  
 console.log(json);  
});

Y bueno, para ustedes en este momento además de la legibilidad del fetch, el XMLHTTPRequest con menos líneas hace el mismo resultado… ¿o no?

**Nota:** Para usar estos ejemplos debemos usar el navegador.

Vamos a ejecutar esto así:

<!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>Fetch API</title>  
</head>  
<body>  
 <script>  
 console.log("Antes del request");  
 var req = new XMLHttpRequest();  
 req.open('GET', 'http://127.0.0.1:8887/datasets/westeros.json', false);  
 req.send(null);  
 if (req.status == 200) {  
 console.log(JSON.parse(req.responseText));  
 }  
 console.log("Después of request");  
   
   
 console.log("Antes request con fetch");  
 fetch(`http://127.0.0.1:8887/datasets/westeros.json`,{})  
 .then(function(response) {  
 return response.json();  
 })  
 .then(function(json) {  
 console.log(json);  
 });  
 console.log("Después del request con fetch");  
  
 </script>  
</body>  
</html>

### 1.1 Promesas (Promises)

El objeto Promise representa la finalización (o el fallo) de una operación asíncrona, y su valor resultante. – MDN Mozilla –

Las promesas funcionan para ejecutar peticiones u operaciones de manera asincrona, retornando un objeto que en el tiempo se resolverá y al hacerlo ejecutará lo que se requiera.

#### Creación de una promesa:

var promise = new Promise(function(resolve, reject) {  
 setTimeout(function() {  
 resolve('foo');  
 }, 300);  
 });  
   
 promise.then(function(value) {  
 console.log(value);  
 // expected output: "foo"  
 });  
   
 console.log(promise);  
 // expected output: [object Promise]

Así mismo al ejecutar el método then recibimos como parámetro el valor retornado en la respuesta, que en algunos casos podrá ser también una promesa como fue el caso anterior con fetch.