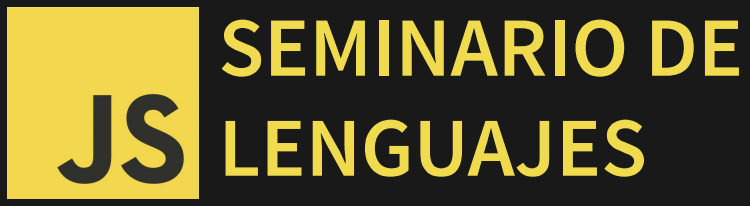

CLASE 6 - API FETCH

ASINCRONISMO EN JAVASCRIPT - API FETCH

Gestión de asincronismo en JavaScript - API Fetch

AGENDA

1. Web Api
2. Api Fetch1
3. Api Fetch2
4. Fetch - Parametros1
5. Fetch - Parametros2
6. Fetch - Parámetros - Opciones
7. El Objeto Response
8. El Objeto Response. Métodos
9. Objeto Response
10. Objeto Request
11. Referencias

WEB API

Las Web APIs conectan aplicaciones con servicios de la Web. Son muy útiles para crear aplicaciones complejas.

Permiten conectar a servidores y clientes Web a través de una interfaz común. En JavaScript existen APIs del lado del cliente y del lado del servidor. Por ejemplo los servicios de geolocalización, la API de Twitter o de Google.

Del lado del servidor puede ser una aplicación móvil que se conecta al servidor del servicio meteorológico y muestra información actualizada.

API FETCH[1]

Proporciona una interfaz JavaScript para manipular partes del canal HTTP relacionado con peticiones y respuestas.

El método fetch() permite recuperar recursos en forma asincrónica a través de la red.

Permite asociar otros conceptos relacionados con HTTP como CORS y extensiones HTTP.

API FETCH[2]

```
fetch("https://mdn.github.io/learning-area/javascript/oc
  .then(response => response.text())  // toma la respues
  .then(data => console.log(data))  // gestiona el objet
  .catch(() => console.log("Error!"))
  .finally(() => console.log("Terminado!"))
```

Obtenemos un archivo json de la red y lo mostramos en consola. No se rechaza con un código de error 500 o 404 sino con mensaje de error.

FETCH - PARAMETROS[1]

fetch(url, options)

- URL: obligatorio
- method: string. Método de la petición HTTP. Por defecto GET.
- headers: objeto. Cabeceras HTTP. Por defecto {}, es posible indicar el Content-type y el encoding por ejemplo.
- body: cuerpo de la petición HTTP. Puede ser de distinto tipo, como String, Blob, etc.

FETCH - PARAMETROS²

FETCH - PARÁMETROS - OPCIONES

```
//  Opciones de la petición
const opciones = {
    method: "GET"
};

//Petición HTTP
fetch("http://misitio.edu.ar/gatitos.json", opciones)
    .then(response => response.text())
    .then(data => {
        /** Procesar los datos **/
});
```

```
//  Opciones de la petición
const opciones = {
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    body: JSON.stringify(jsonData)
};


//Petición HTTP
fetch("http://misitio.edu.ar/gatitos.json", opciones)
    .then(response => response.text())
```

EL OBJETO RESPONSE

```
//En forma más eficiente
fetch('http://misitio.com.ar/ajax-info.json')
  .then(function(response)){
    /** Código que procesa la respuesta **/
    });
```

- Objeto Response. Posee propiedades y métodos.

Propiedad	Descripción	Tipo
.status	Código de error HTTP de la respuesta (100-599).	Number
.statusText	Texto representativo del código de error HTTP anterior.	String
.ok	Devuelve true si el código HTTP es 200 (o empieza por 2).	Boolean
.headers	Cabeceras de la respuesta.	Objeto
.url	URL de la petición HTTP.	String

EL OBJETO RESPONSE. MÉTODOS

```
fetch('http://misitio.com.ar/ajax-info.json')
 .then(response => response.text())
 .then(data => console.log(data));
```

Método	Descripción	Tipo
.text()	Devuelve una promesa con el texto plano de la respuesta.	String
.json()	Idem anterior con un objeto json. Equivalente a usar JSON.parse().	Objeto
.blob()	Idem anterior con un objeto Blob (binary large object).	Objeto
.arrayBuffer()	Idem anterior con un objeto ArrayBuffer.	Objeto
.formData()	Idem anterior con un objeto FormData.	Objeto
.clone()	Crea y devuelve un clon de la instancia.	Objeto
Response.error()	Devuelve un nuevo objeto Response con un error de red asociado.	Objeto
Response.redirect(url, code)	Redirige a una url, opcionalmente con un code de error.	Objeto

OBJETO RESPONSE

```
// Petición HTTP
fetch('http://misitio.com.ar/ajax-info.json')
  .then(response => {
    if (response.ok)
        return response.text()
    else
        throw new Error(response.status);
    })
.then(data => {
    console.log("Datos: " + data);
 })
.catch(err => {
    console.error("ERROR: ", err.message)
});
```

Comprobamos que la petición es correcta.

Procesamos la respuesta con .text()

Si hubo error, levantamos una excepción con el código de error.

Procesamos los datos y se muestran en consola.

Si la promise es rechazada, se captura el error con catch().

```
// Alternativa a la anterior
const isResponseOk = (response) => {
if (!response.ok)
    throw new Error(response.status);
    return response.text()
}

fetch('http://misitio.com.ar/ajax-info.json')
  .then(response => isResponseOk(response))
  .then(data => console.log("Datos: ", data))
  .catch(err => console.error("ERROR: ", err.message));
```

- Se agrega una función flecha para procesar la respuesta.
- Los .then y .data se utilizan también con funciones flecha para simplificarlos.

OBJETO REQUEST

- JS permite crear nuestros propios objetos request, constructor Request.
- Se puede utilizar para pasar como parámetro a 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;
});
```

REFERENCIAS

JavaScript from Frontend to Backend. Eric Sarrion. Ed. Packt Publishing (2022)

MDN Guía de JavaScript. Usar promesas

JavaScript Promises. W3Schools

Callbacks y Promises | Explicado con ejemplos

Así funcionan las PROMESAS y ASYNC/AWAIT en JAVASCRIPT.