sustantiva

CON•SENTIDO

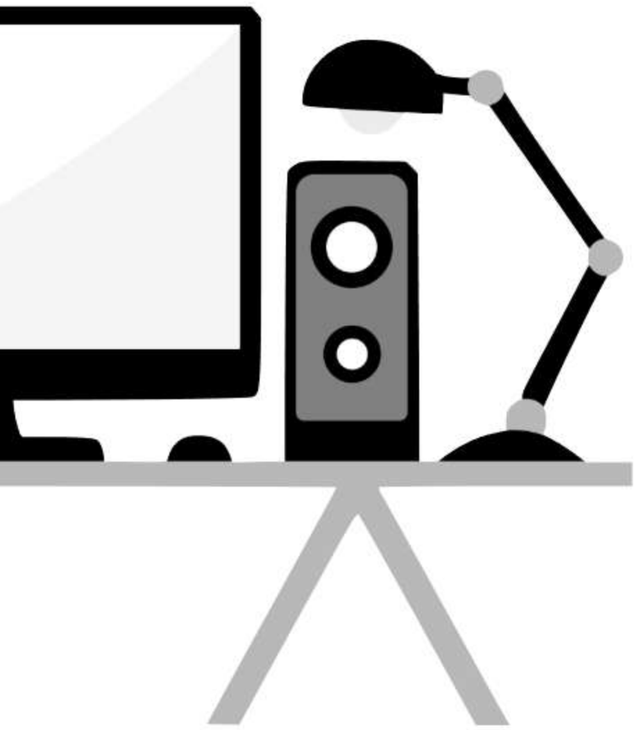
CONSUMO DE APIs CON JAVASCRIPT

Lección 05

Competencia General

Utilizar el objeto XHR y la API Fetch para el consumo de una API externa y su procesamiento acorde al lenguaje JavaScript.

Contenidos



Apis en JavaScript

1. Qué es una API
2. APIs de JavaScript del lado del cliente
3. APIs del browser
4. APIs de terceros

Obtención de data desde un servidor

1. Ajax
2. APIs XHR y Fetch
3. Realizando request con XHR
4. Realizando request con Fetch

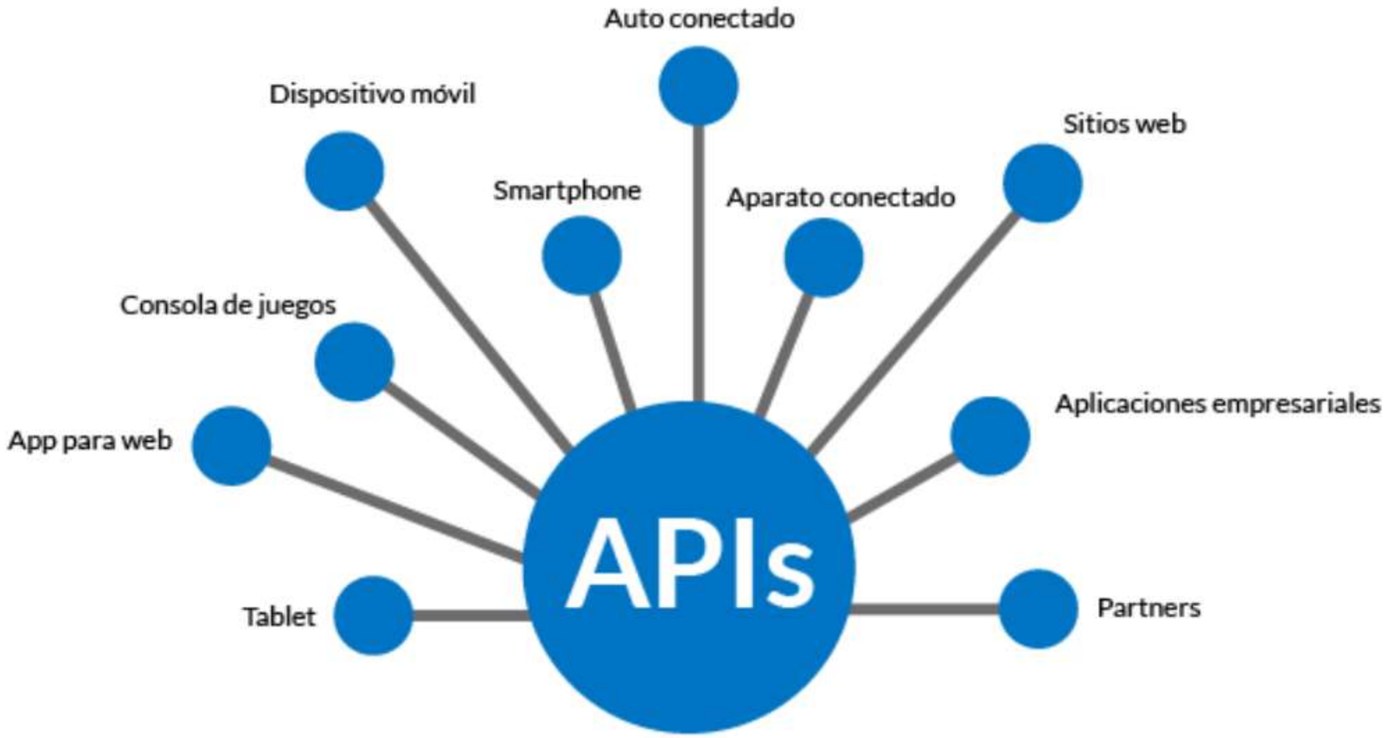
APIs de terceros

1. Solicitando data a una API
2. Procesando la respuesta

¿Qué es una API?

Una API (Application Programming Interfaces o interfaz de programación de aplicaciones) es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar software, permitiendo la comunicación entre dos aplicaciones a través de un conjunto de reglas.

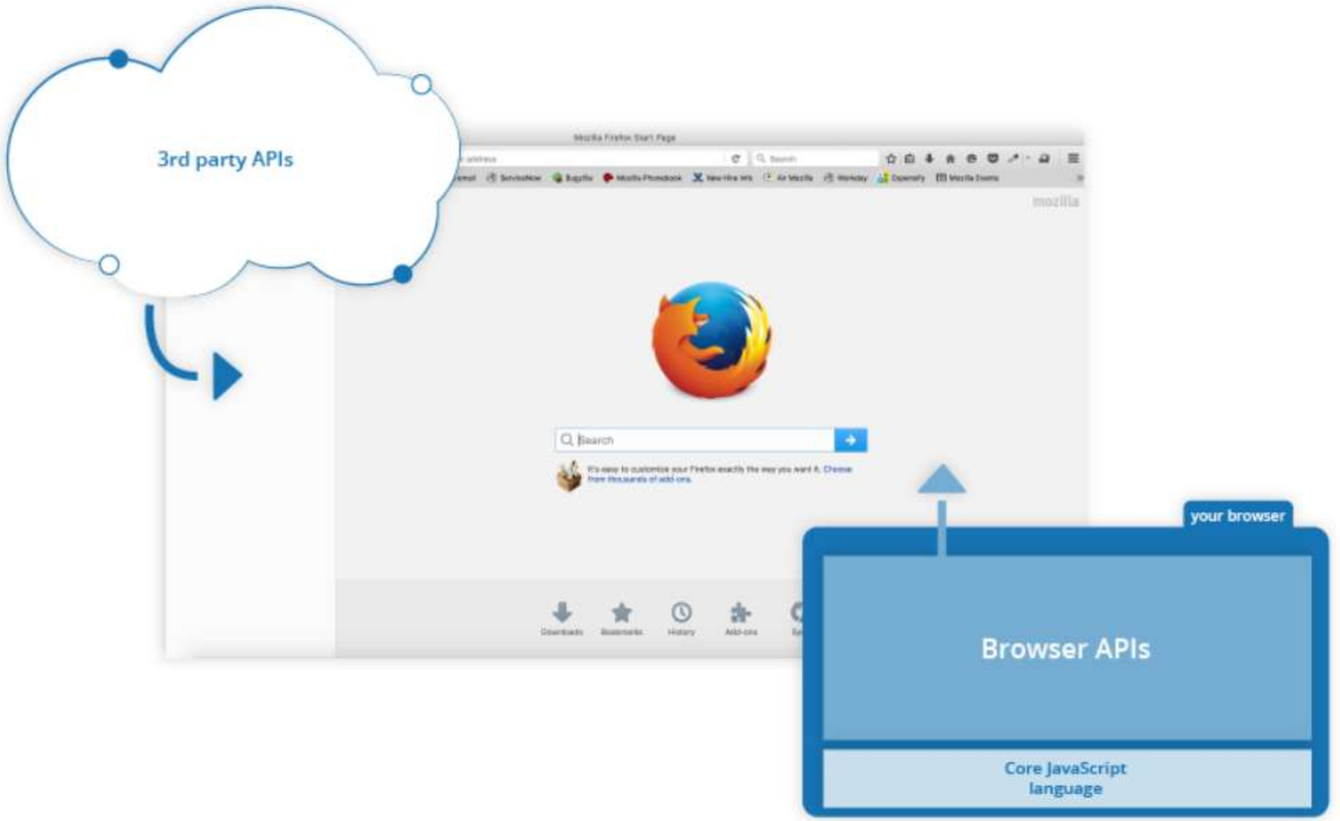
En otras palabras se trata de como establecen comunicación e interactúan dos aplicaciones.



APIs del lado del cliente

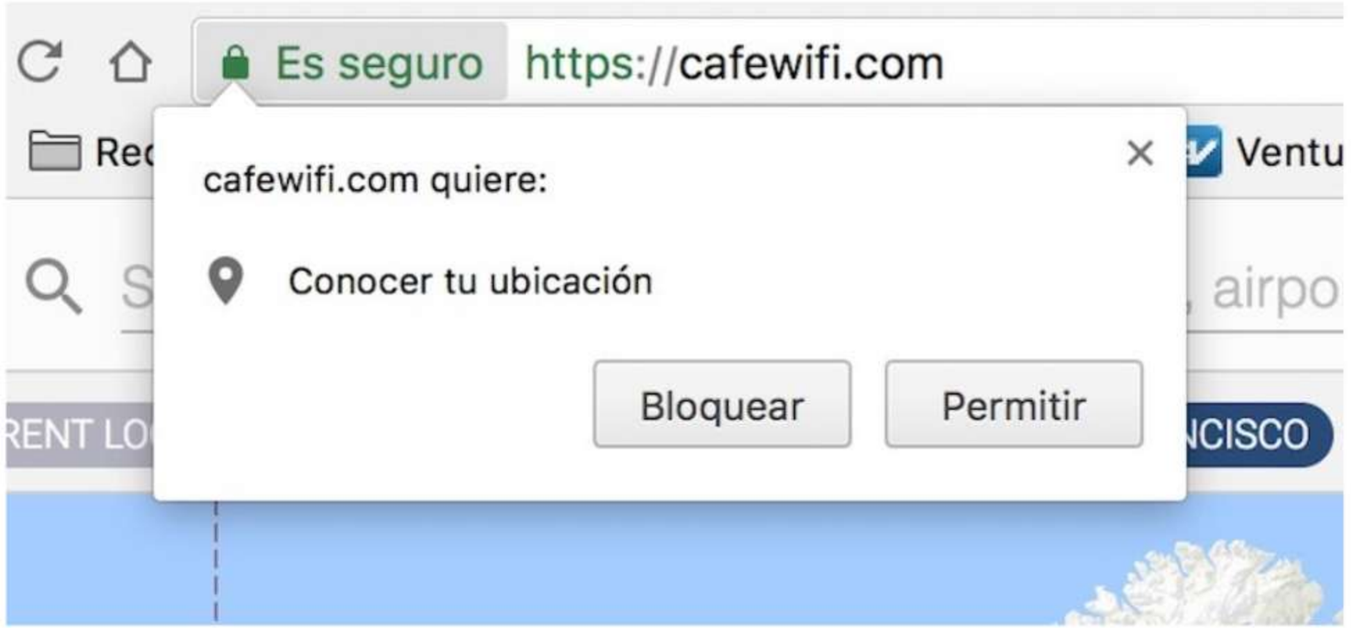
JavaScript del lado cliente, particularmente, tiene muchas APIs disponibles, estas no son parte del lenguaje en sí, sino que están construidas sobre el núcleo de este lenguaje de programación, proporcionándote superpoderes adicionales para usar en tu código. Por lo general, se dividen en dos categorías:

- APIs del navegador
- APIs de terceros



APIs del navegador

Las APIs de navegador están integradas en tu navegador web y pueden exponer datos del navegador y del entorno informático circundante y hacer cosas complejas y útiles con él. Por ejemplo, la API de Geolocalización proporciona algunas construcciones simples de JavaScript para obtener datos de ubicación con los que, por ejemplo, trazar tu ubicación en un mapa de Google.



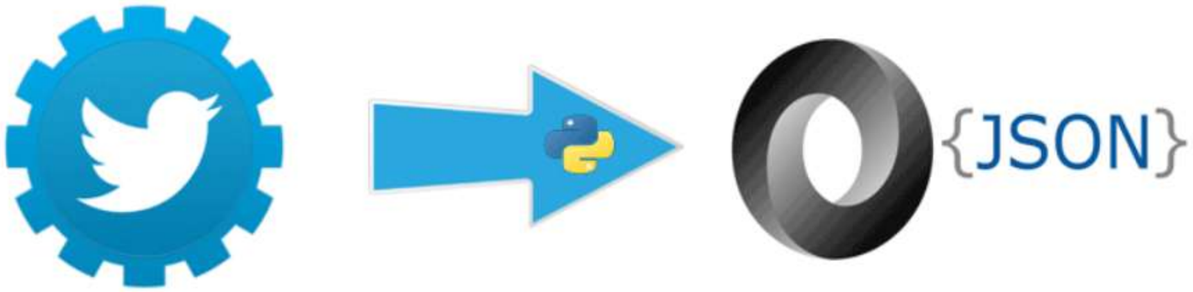
APIs del navegador

Las APIs del navegador más comunes:

1. APIs para manipular el DOM que permite manipular HTML y CSS. Crear, eliminar y modificar HTML, aplicar estilos dinámicos a una página, etc.
2. APIs para dibujar y manipular gráficos como Canvas y WebGL para crear escenas 2D y 3D. Por ejemplo, se pueden dibujar formas como rectángulos o círculos, importar una imagen en el canvas y aplicarle filtros como sepia o escala de grises usando la API de Canvas, o crear una escena compleja 3D con iluminación y texturas usando WebGL.
3. APIs de audio y video como WebRTC o HTMLMediaElement te permiten crear una interfaz personalizada para los controles de reproducción de audio y vídeo, mostrar pistas de texto con subtítulos junto con el vídeo, capturar vídeo de la cámara web para ser manipulado en un canvas

APIs de terceros

Las APIs de terceros no están incluidas por defecto en el navegador, y por lo general es necesario obtener el código e información desde algún lugar de la Web. Por ejemplo, la API de Twitter permite hacer cosas como mostrar tus últimos tweets en un sitio web. Proporciona un conjunto especial de construcciones que puedes usar para consultar el servicio de Twitter y devolver información específica.



APIs de terceros

APIs de terceros comunes:

1. API de Twitter (https://dev.twitter.com/overview/documentation), te permite hacer cosas como mostrar tus ultimos tweets en tu sitio web.
2. API de Google Maps (https://developers.google.com/maps/), te permite hacer todo tipo de cosas con mapas en tus páginas web (incluso hace funcionar Google Maps).
3. API de Youtube (https://developers.google.com/youtube/), permite hacer todo tipo de cosas con mapas en tus páginas web (incluso hace funcionar Google Maps).

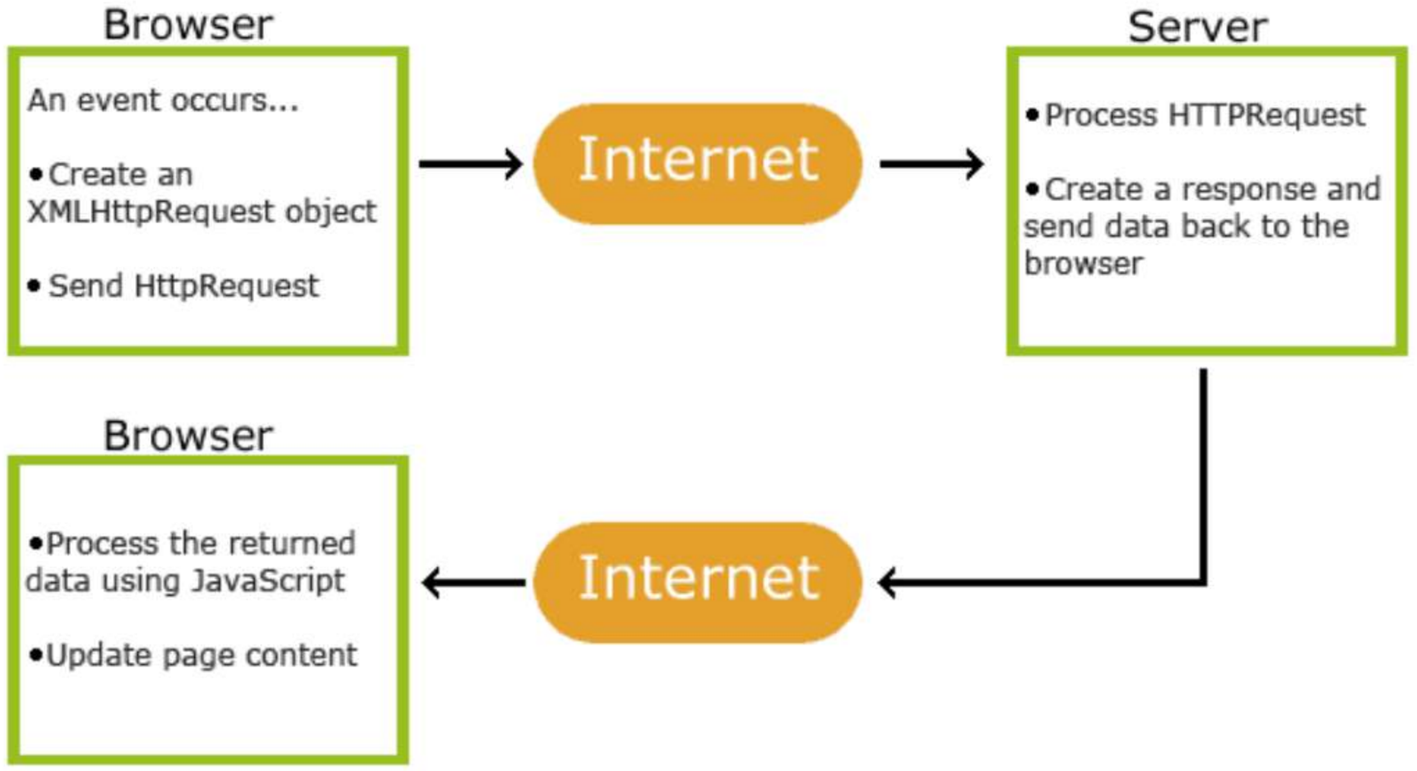
AJAX

AJAX (Asynchronous JavaScript And XML) es una técnica que combina varias tecnologías web para permitir la **comunicación asíncrona entre el navegador y el servidor**, sin necesidad de recargar toda la página.

Se basa en:

1. **XMLHttpRequest (o Fetch API):** Permite enviar y recibir datos del servidor de forma asíncrona.

2. **JavaScript y el DOM:** Para procesar la respuesta y actualizar sólo partes específicas de la página.



AJAX

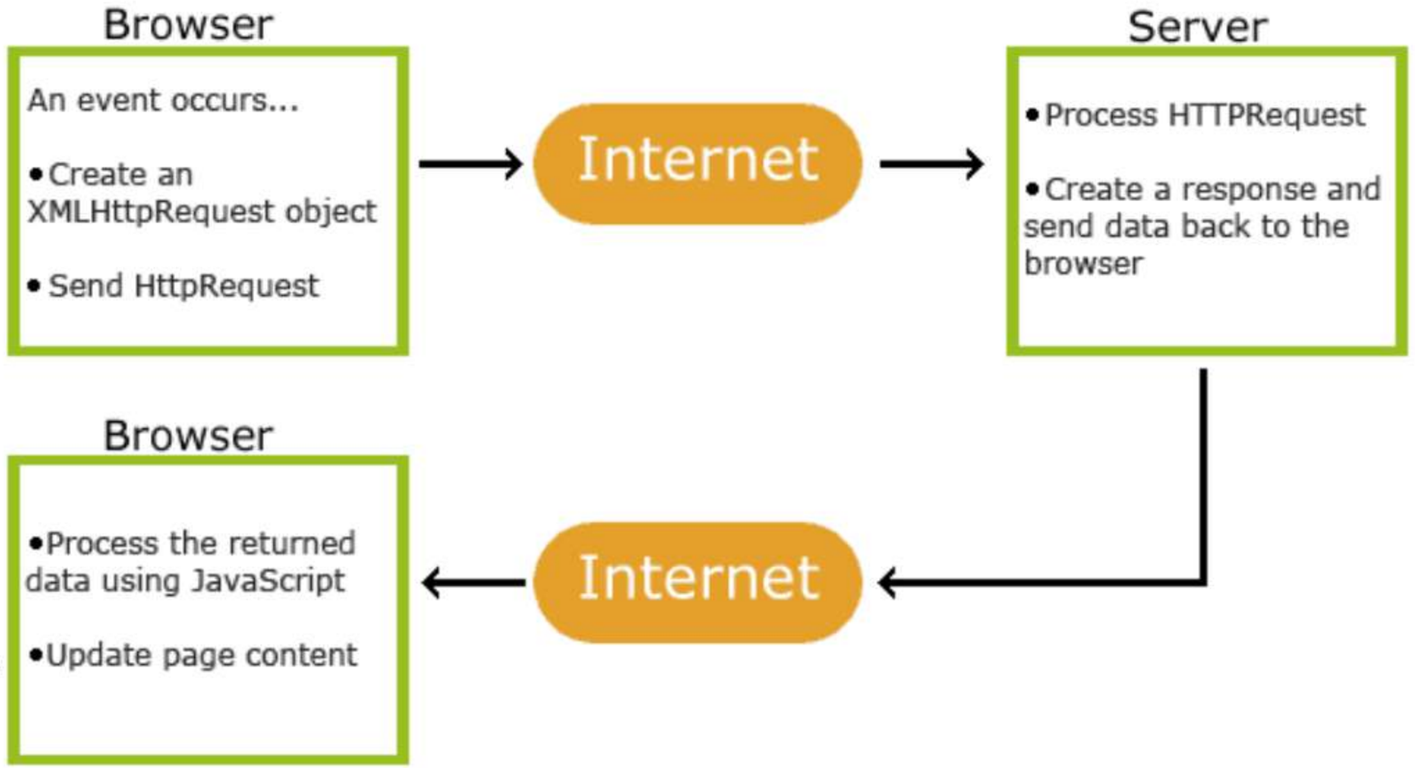
¿Qué problema resuelve?

Antes de AJAX, cada vez que el usuario interactuaba con una página (por ejemplo, enviar un formulario), era necesario **recargar la página completa** para mostrar los nuevos datos. Esto generaba:

- Una experiencia de usuario lenta y poco fluida.
- Mayor consumo de ancho de banda, ya que se descargaba toda la página.

AJAX permite:

- Actualizar secciones de una página sin recargarla.
- Mejorar la experiencia del usuario con interfaces más dinámicas.
- Reducir la carga en el servidor y el consumo de datos.



XHR y Fetch

Ambos son métodos que nos retornan información desde el servidor usando AJAX. Sin embargo cada una tiene sus pro y contras.

Fetch	XHR
No existe forma de establecer un tiempo de espera	No se pueden enviar cookies
No hay progreso de carga	No tiene un equivalente al modo no-CORS
No se puede invalidar el encabezado de una respuesta	Puedes abortar una solicitud
Puedes realizar peticiones a un servidor que no implementa CORS	Puedes informar sobre el progreso de una solicitud
Puedes obtener respuesta de streaming (transmisión a bajo nivel)	

Request con XHR

Para enviar una petición con XHR debemos:

- **Crear el objeto** XMLHttpRequest().
- **Abrir la solicitud** con el método open(método, url, asíncrono) indicando:
  - El **método HTTP** (GET, POST, etc.).
  - La **URL** del recurso.
  - Si será asíncrona (true) o síncrona (false). *(Generalmente se usa true)*.
- **Definir la función que manejará la respuesta**, observando la propiedad readyState (debe ser 4 cuando finaliza) y verificando el código de estado status (debe ser 200 para éxito).
- **Enviar la petición** con send().

```
var req = new XMLHttpRequest();
req.open('GET', 'URL', true);

req.onreadystatechange = function () {
  if (req.readyState == 4) {
    if (req.status == 200) {
      console.log("Éxito: ", req.responseText);
    } else {
      console.log("Error: ", req.status);
    }
  }
};

req.send();
```

Request con Fetch

La API de **Fetch** es más limpia y fácil de entender que XHR, ya que está basada en **promesas**.

- Debemos crear la solicitud fetch() indicando la **URL** o recurso.
- El **primer then()** procesa la respuesta de la promesa. Si la petición se resuelve correctamente, podemos transformar el contenido (por ejemplo, a JSON).
- El **segundo then()** gestiona los datos ya transformados.
- El método **catch()** se utiliza para gestionar errores o rechazos de la petición.

```
fetch(url, options)
  // Transforma la respuesta a JSON
  .then((resp) => resp.json())
  .then((data) => {
    let datos = data.results;
    console.log(datos);
  })
  .catch((error) => {
    console.log("Error:", error);
  });
```

Cuándo usar XHR o Fetch

Usar XHR:

- Cuando se necesita compatibilidad con navegadores antiguos (ej.: Internet Explorer).
- Si es necesario un control detallado del progreso de la petición (cómo mostrar barras de carga).

Usar Fetch:

- En la mayoría de los proyectos actuales, ya que es moderno y más sencillo de usar.
- Permite trabajar con promesas de forma nativa, haciendo el código más limpio y fácil de mantener.

API-KEY

Existen APIs de terceros que tienen sus datos protegidos con usuario y contraseña, para acceder a estos datos es necesario enviar en las peticiones una “api-key”.
Una API Key no es más que un identificador (una clave y contraseña para autenticarte), y al cual se le asignarán permisos para acceder a la información.
Para incluir esta información debemos enviar en los headers de una petición la api-key

```
headers: {
  'X-API-KEY': 'apikey',
  'Accept': 'application/json',
  'Content-Type': 'application/json'
 }
```

API-KEY

Este código muestra cómo consumir una API que **requiere una API Key** para funcionar. En este ejemplo, se utiliza **The Cat API**, la cual entrega imágenes aleatorias de gatos. Por razones de seguridad, se ha dejado una clave de acceso simulada (DEMO-API-KEY) en lugar de una real, ya que las API Keys son **privadas y no deben compartirse públicamente**.

```
const url = 'https://api.thecatapi.com/v1/images/search';
const apiKey = 'DEMO-API-KEY'; // tu API-KEY real

function obtenerGatito() {
  fetch(url, {
    headers: {
      'x-api-key': apiKey,                // API KEY
      'Accept': 'application/json',       // formato de respuesta
      'Content-Type': 'application/json' // formato de envío
    }
  })
  .then(response => response.json())
  .then(data => {
    const imagenUrl = data[0].url; // la imagen del gato viene en data[0].url
    document.getElementById('catImage').src = imagenUrl;
  })
  .catch(error => {
    console.error('Error al obtener el gatito:', error);
  });
}
obtenerGatito();
```

Gatito con API-KEY

Cargar otro gatito

Cómo funciona el código:

- **url**: Corresponde al endpoint de la API, que en este caso entrega imágenes aleatorias de gatos.
- **apiKey**: Es la clave única proporcionada al registrarse en The Cat API. Esta clave debe incluirse en los headers de la petición para autenticar el acceso.
- **fetch()**: Se utiliza para realizar una solicitud HTTP al servidor de la API.
  - Dentro del fetch, se configura el objeto headers con:
    - 'x-api-key': contiene el valor de la API Key.
    - 'Accept' y 'Content-Type': indican que se espera y se envía contenido en formato JSON.
- **.then()**: Encadena promesas que se ejecutan una vez que la solicitud ha sido exitosa.
  - Primero transforma la respuesta a formato JSON.
  - Luego accede a la URL de la imagen en data[0].url y la inserta dinámicamente en el elemento HTML con ID "catImage".
- **.catch()**: Captura errores en caso de que la solicitud falle, mostrando un mensaje en la consola.

Solicitando data

Ejemplo práctico:

Vamos a solicitar datos desde una api de usuarios:

1. Guardamos la URL del servidor en una variable

```
const url = 'https://randomuser.me/api/?results=10';
```

2. Creamos la petición fetch con los métodos then y catch

```
fetch(url)
.then()
.catch();
```

Solicitando data

3. Transformamos la respuesta del servidor a json

```
fetch(url)
.then((resp) => resp.json())
.catch();
```

4. Crea una nueva promesa con un then encadenado que permita guardar los datos transformados a json.

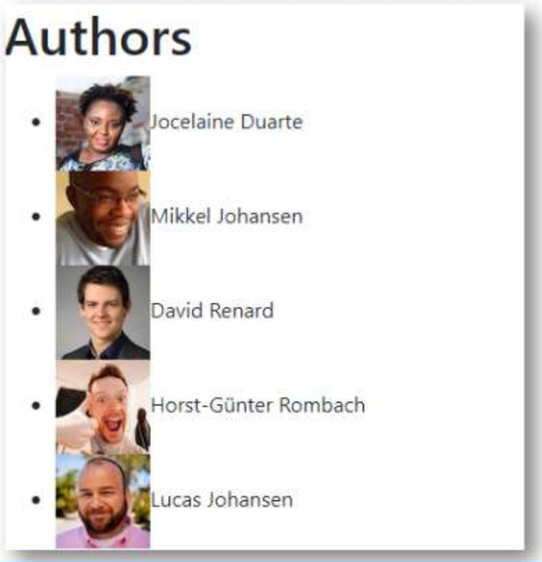
```
fetch(url)
.then((resp) => resp.json())
.then(function(data) {
  let authors = data.results;
  console.log(authors)
})
.catch();
```

Solicitando data

5. Visualiza los datos recibidos en la consola del navegador

Actividad propuesta.

Utiliza estos datos para mostrar la información en una página. Para ello ayúdate con la lección anterior donde se insertan elementos al DOM. El resultado debe ser similar a este:



Solicitando data

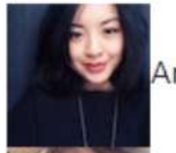
Solución:

```
const ul = document.getElementById('authors');
const url = 'https://randomuser.me/api/?results=10';
fetch(url)
.then((resp) => resp.json())
.then(function(data) {
  let authors = data.results;
  return authors.map(function(author) {
    let li = document.createElement('li');
    let img = document.createElement('img');
    let span = document.createElement('span');
    img.src = author.picture.medium;
    span.innerHTML = `${author.name.first} ${author.name.last}`;
    li.appendChild(img)
    li.appendChild(span)
    ul.appendChild(li)
  })
})
.catch(function(error) {
  console.log(error);
});
```

Solicitando data

Solución:

Authors

-  Angie Alexander
-  Indie Lee
-  Luis Gibson
-  Brendan Fitzgerald
-  Keith Hawkins
-  Yolanda Jordan


sustantiva
CON•SENTIDO