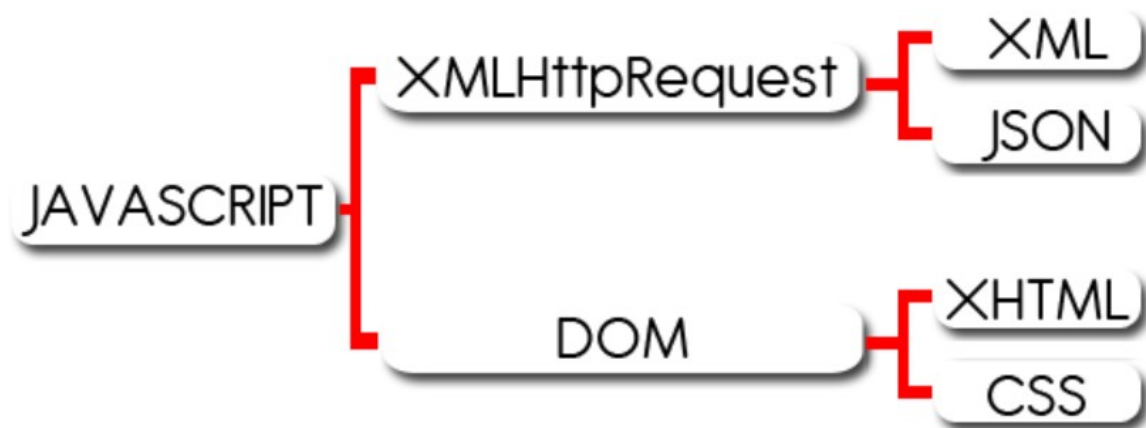


**AJAX: Asynchronous Javascript And XML” (Javascript asíncrono y XML).**

AJAX en si no es una tecnología, sino un conjunto de tecnologías. Permite comunicarse con el servidor, intercambiar datos y actualizar la página **sin tener que recargar el navegador**.

Las tecnologías presentes en AJAX son:

- XHTML y CSS para la presentación de la página.
- DOM para la manipulación dinámica de elementos de la página.
- Formatos de intercambio de información como JSON o XML.
- El objeto XMLHttpRequest, para el intercambio asíncrono de información (es decir, sin recargar la página).
- Javascript, para aplicar las anteriores tecnologías.



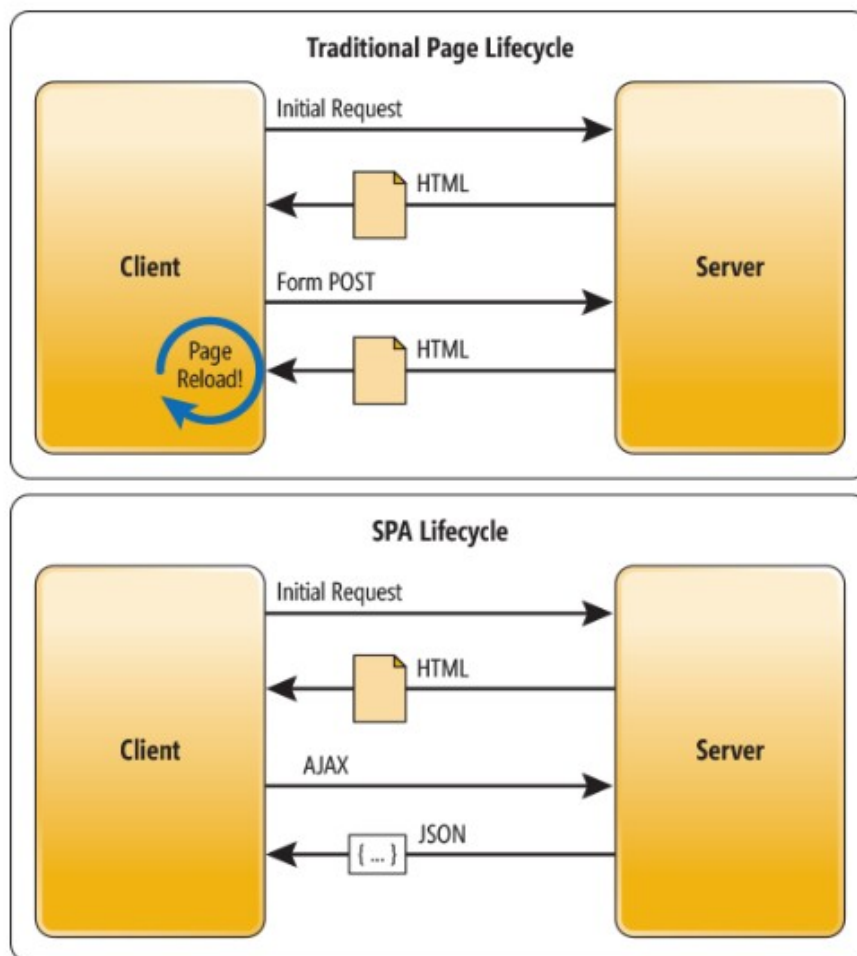
La forma de trabajar es la siguiente: JavaScript se encarga de unir todas las tecnologías. Para manipular la parte de representación de la página utiliza DOM (manipula el XHTML y CSS)..

Para realizar peticiones asíncronas usa el objeto XMLHttpRequest. Este objeto intercambia información que son simplemente cadenas de texto. Lo más habitual es utilizar JSON o XML.

Como resultado obtenemos una navegación ágil, rápida y dinámica; y también la posibilidad de realizar cambios sobre una web sin necesidad de actualizarla.

Cuando interactuamos con servidores, podemos hacer uso de diferentes métodos HTTP para solicitar datos. Podemos crear, leer, actualizar y eliminar (CRUD) datos en los servidores utilizando verbos HTTP específicos como POST, GET, PUT/PATCH y DELETE.

- **GET:** Leer.
- **POST:** Crear.
- **PUT:** Actualizar.
- **DELETE:** Borrar.

**Web tradicional vs Ajax:**

En una aplicación Web clásica:

1. El cliente hace una petición al servidor.
2. El servidor recibe la petición.
3. El servidor procesa la petición y genera una nueva página con la petición procesada. (Ejemplo, se añade un post a un foro).
4. El cliente recibe la nueva página completa y la muestra.

En una aplicación Web AJAX:

1. El cliente hace una petición asíncrona al servidor.
2. El servidor recibe la petición.
3. El servidor procesa la petición y responde asíncronamente al cliente.
4. El cliente recibe la respuesta y con ella modifica dinámicamente los elementos afectados de la página sin recargar completamente la página.

Las aplicaciones Web AJAX son mejores ya que reducen la cantidad de información a intercambiar (no se envía la página entera, sino que se modifica solo lo que interesa) y a su vez al usuario final le da una imagen de mayor dinamismo, viendo una página web como una aplicación de escritorio.

El cliente es el programa que envía una solicitud, mientras que el servidor es el que recibe la solicitud. El servidor devuelve una respuesta en función de la validez de la solicitud. Si la solicitud tiene éxito, el servidor devuelve los datos en formato XML o JSON (JSON en la mayoría de los casos), y si la solicitud falla, el servidor devuelve un [mensaje de error](#).

Las respuestas que devuelve el servidor suelen estar asociadas a [códigos de estado \(status\)](#). Estos códigos nos ayudan a entender lo que el servidor intenta decir cuando recibe una petición. Aquí tienes algunos de ellos y su significado:

- 100-199 denota una respuesta informativa.
- 200-299 **denota una solicitud exitosa.**
- 300-399 denota una redirección.
- 400-499 indica un error del cliente.
- 500-599 denota un error del servidor.

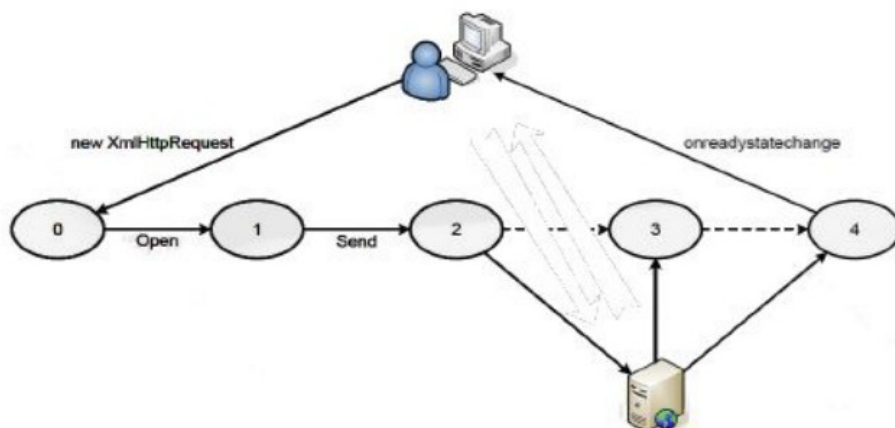
### Métodos JS Nativo:

- XMLHttpRequest
- API Fetch

### Librerías externas

- JQuery Ajax
- Axios
- ...

### Objeto XMLHttpRequest



¿Cómo lo aplicamos en código? El evento `onreadystatechange` cada vez que se produzca comprobará en que estado nos encontramos y hará lo planeado para ese estado. Generalmente el estado más utilizado es el 4, donde se ha completado la operación.

**Códigos de respuesta (readyState) en JavaScript que indican el proceso de una solicitud:**

- 0 al inicializarse el objeto ( pero aun no se ha llamado)
- 1 al abrirse una conexión (al usar el método open).
- 2 al hacer una petición (uso de send).
- 3 mientras se está recibiendo información de la petición.
- 4 cuando la petición se ha completado.

Estado	Valor
<i>READY_STATE_UNINITIALIZED</i>	0
<i>READY_STATE_LOADING</i>	1
<i>READY_STATE_LOADED</i>	2
<i>READY_STATE_INTERACTIVE</i>	3
<i>READY_STATE_COMPLETE</i>	4

**Cómo enviar una solicitud GET en JavaScript utilizando XMLHttpRequest**

Utilizas la petición GET cuando quieres recuperar datos de un servidor. Para enviar una solicitud GET con éxito utilizando XMLHttpRequest en JavaScript, debes asegurarte de que lo siguiente se hace correctamente:

1. Crea un nuevo objeto XMLHttpRequest.
2. Abre una conexión especificando el tipo de petición y el punto final (la URL del servidor).
3. Envía la petición.
4. Espera la respuesta del servidor.

```
const xhr = new XMLHttpRequest();
xhr.open("GET", "https://jsonplaceholder.typicode.com/users");
xhr.send();

xhr.addEventListener("readystatechange", (e) => {
  if (xhr.readyState !== 4) return;

  if (xhr.status >= 200 && xhr.status < 300) {
    console.log("éxito");

    let json = JSON.parse(xhr.responseText);
    console.log(json);

    json.forEach((el) => {
      const $li = document.createElement("li");
      $li.innerHTML = `${el.name} -- ${el.email} -- ${el.phone}`;
      $fragment.appendChild($li);
    });

    $xhr.appendChild($fragment);
  } else {
    console.log("error");
    let message = xhr.statusText || "Ocurrió un error";
    $xhr.innerHTML = `Error ${xhr.status}: ${message}`;
  }

  console.log("Este mensaje cargará de cualquier forma");
});
```

**Cómo enviar una petición POST en JavaScript utilizando XMLHttpRequest**

```
const xhr = new XMLHttpRequest();
xhr.open("POST", "https://jsonplaceholder.typicode.com/posts");

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

const body = JSON.stringify({
  title: "Hello World",
  body: "My POST request",
  userId: 900,
});

xhr.addEventListener("readystatechange", (e) => {

  if (xhr.readyState !== 4) return;

  if (xhr.status >= 200 && xhr.status < 300) {
    console.log(JSON.parse(xhr.responseText));
  } else {
    console.log(`Error: ${xhr.status}`);
  }
});
xhr.send(body);
```

**Cómo enviar una petición PUT en JavaScript utilizando XMLHttpRequest**

```
const xhr = new XMLHttpRequest();
xhr.open("PUT", "https://jsonplaceholder.typicode.com/posts");

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

const body = JSON.stringify({
  title: "Hello World",
  body: "My POST request",
  userId: 7,
});

xhr.addEventListener("readystatechange", (e) => {
  if (xhr.readyState !== 4) return;
  if (xhr.status >= 200 && xhr.status < 300) {
    console.log(JSON.parse(xhr.responseText));
  } else {
    console.log(`Error: ${xhr.status}`);
  }
});

xhr.send(body);
```

La información que se enviará al servidor se almacena en una variable llamada `body`. Contiene tres propiedades: `title`, `body` y `userId`.

Ten en cuenta que la variable `body` que contiene el objeto debe ser convertida en un objeto JSON antes de ser enviada al servidor. La conversión se realiza mediante el método **`JSON.stringify()`**.

Para asegurarte de que el objeto JSON se envía al servidor, se pasa como parámetro al método `send()`.

### Cómo enviar una solicitud DELETE en JavaScript utilizando XMLHttpRequest

```
const xhr = new XMLHttpRequest();
xhr.open("DELETE", "https://jsonplaceholder.typicode.com/posts/3");
xhr.addEventListener("readystatechange", (e) => {
  var data = JSON.parse(xhr.responseText);
  if (xhr.status >= 200 && xhr.status < 300) {
    console.log(data);
  } else {
    console.log(`Error: ${xhr.status}`);
  }
});
xhr.send();
```

#### **Importante:**

**JSON.parse:** se utiliza para convertir una cadena de texto con formato JSON en un objeto o valor de JavaScript. Ejemplo:

```
const jsonString = '{"name": "Juan", "age": 30, "isDeveloper": true}';
const obj = JSON.parse(jsonString);
console.log(obj); // { name: "Juan", age: 30, isDeveloper: true }
console.log(obj.name); // "Juan"
console.log(obj.age); // 30
```

**JSON.stringify:** convierte un valor en JavaScript (como un objeto, arreglo, número, etc.), en una cadena de texto en formato JSON. Es lo opuesto a `JSON.parse`.

Ejemplo:

```
const arr = [1, 2, 3, "cuatro", { key: "valor" }];
const jsonString = JSON.stringify(arr);
console.log(jsonString); // '[1,2,3,"cuatro",{"key":"valor"}]'
```

**Ejercicios:**

1.- Obtener el usuario con id=5 de [jsonplaceholder.typicode.com/users](https://jsonplaceholder.typicode.com/users).

Mostrar los datos del usuario en distintos <p> de un div de una página html: Nombre, usuario, correo y la dirección.

2.- Crear una página html con un <H1> POSTS </H1> y un botón. Cuando pulsemos un botón vamos a llamar a la api: [jsonplaceholder.typicode.com/posts](https://jsonplaceholder.typicode.com/posts) , y vamos a pintar el resultado en una **tabla**. Sacar en cada fila el title y el body, cada uno en una columna diferente.

3.- Crear una página html con un <h1> ¿Sí o No?. Tendrá un botón y un p para mostrar ahí la respuesta.

Cuando pulsemos el botón, se conectará con la api [yesno.wtf/api](https://yesno.wtf/api) para obtener la respuesta Si o No.

Crear otra página html2 igual a la anterior, tal que al pulsar el botón, cargaremos en el resultado la imagen que devuelva.

4.- Dada la api: [opentdb.com/api.php?amount=5&type=multiple](https://opentdb.com/api.php?amount=5&type=multiple), mostrar una lista con todas las pistas que aparecen: Category, question y correct\_answer. La etiqueta ul está creada en el html, las li no.

Estilos para los li (poner en js): backgroundColor: #e9e9e9, padding: 10px, margin: 5px 0, border: 1px solid #ccc, borderRadius: 4px, fontFamily: Arial, sans-serif y color: #333.

Poner la correct\_answer en negrita.

Poner también que al pasar por encima de alguna li, el color de fondo sea: #d1d1d1

5.- Crear un fichero json en vuestro proyecto con nombre data.json. Crear un objeto literal students que será un array de objetos literales. Dentro de cada objeto habrá información de cada alumno: id: numero, nombre:cadena y notas, que será un array de 4 notas. Crear datos para 5 alumnos en el fichero.

A partir de este html:

```
<table id="studentTable">
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Grades</th>
      <th>Average</th>
    </tr>
  </thead>
  <tbody></tbody>
</table>
```

Crear un fichero js que haga una llamada ajax a dicho fichero y crear una tabla con la información de los alumnos con 4 columnas: ID , nombre, notas, que aparecerán separadas por comas y la última columna será la media de notas.



## API Fetch

- Es una API nativa de JavaScript.
- Sintaxis básica:

```
fetch(url, options)
  .then(response => {
    // Procesar la respuesta
  })
  .catch(error => {
    // Manejar errores
  });
```

- Fetch utiliza varios parámetros: la url es el primer argumento. El segundo parámetro es opcional y ahí especificamos el cuerpo y el tipo de solicitud.
- Devuelve una **promesa (ES 2015)** para gestionar el código asíncrono. Esto significa que el código no se bloquea mientras espera la respuesta.
  - Una promesa representa la finalización de una función asíncrona.
  - Las promesas disponen:
    - método **then**, que se ejecutará cuando se resuelva la promesa. Recibe como argumento una función anónima que muestra el resultado de la promesa.
    - **Async y await**
- Propiedades importantes:
  - El método fetch devuelve un objeto Response que contiene información sobre la respuesta HTTP:
    - response.ok: true si el código de estado HTTP está en el rango 200-299.
    - response.status: Código de estado HTTP (por ejemplo, 404 o 500).
    - response.json(): Método para convertir la respuesta a formato JSON.
    - response.text(): Método para obtener la respuesta como texto.
    - response.blob(): Método para obtener datos binarios (como imágenes o archivos).

### Ejemplo de Api Fetch GET

```
fetch("https://jsonplaceholder.typicode.com/posts")
  .then(response => response.json()) // CONVIERTE LA RESPUESTA EN UN OBJETO JSON
  .then(data => console.log(data))
  .catch(error => console.log('Error:', error));
```



**Ejemplo de Api Fetch POST**

```
fetch("https://jsonplaceholder.typicode.com/posts",  
      { method: "POST",  
        body: JSON.stringify(  
          { title: "Hello World",  
            body: "My POST request",  
            userId: 900  
          },  
        headers: { "Content-type": "application/json; charset=UTF-8", }  
      })  
.then((response) => response.json())  
.then((json) => console.log(json));
```

**Ejemplo de Api Fetch PUT**

```
fetch("https://jsonplaceholder.typicode.com/posts",  
      { method: "PUT",  
        body: JSON.stringify(  
          { title: "Hello World",  
            body: "Updating request",  
            userId: 900  
          },  
        headers: { "Content-type": "application/json; charset=UTF-8", }  
      })  
.then((response) => response.json())  
.then((json) => console.log(json));
```

**Ejemplo de Api Fetch DELETE**

```
fetch("https://jsonplaceholder.typicode.com/posts/3",  
      { method: "DELETE" });
```

**Ejemplo de Api Fetch async/await get**

Para aportar más legibilidad y facilidad en el manejo del código asíncrono, se utiliza fetch junto con sync/await:

- Se evita así tener que encadenar varios `.then`, ya que se sustituye por `await`.
- Es más fácil de combinar con lógica de control de flujo, como bucles y condiciones.
- Se puede usar `try/catch` para manejar errores en lugar de un `.catch()` adicional

```
async function fetchData() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/posts');
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    const data = await response.json(); // Espera y procesa el JSON
    console.log(data); // Muestra los datos obtenidos
  } catch (error) {
    console.error('Error al realizar la solicitud:', error); // Manejo de errores
  }
}
```

**Ejemplo de Api Fetch async/await put**

```
async function realizarPeticiónPUT(url, datos) {
  const respuesta = await fetch(url, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json', // Asegúrate de ajustar el tipo de contenido según tus necesidades
    },
    body: JSON.stringify(datos), // Convierte tus datos a formato JSON si es necesario
  });

  if (!respuesta.ok) {
    console.log('La petición no fue exitosa');
  }

  const resultado = await respuesta.json(); // Si la respuesta es JSON, puedes parsearla aquí
  console.log('Respuesta exitosa:', resultado); // Puedes hacer algo con la respuesta aquí
}
```








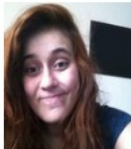


**Ejercicios:**

Repetir los ejercicios 1- 5 con peticiones Fetch con promesas y con async/await.

6.- Insertar un nuevo post en <https://jsonplaceholder.typicode.com/posts>. El campo id no hace falta que lo indiqueis. En html tenéis una etiqueta div para poner el conjunto de datos insertados. Hacer la petición con Fetch y await.

7.- Obtener aleatoriamente datos de un usuario haciendo peticiones ajax sobre la api: <https://randomuser.me/api/>  
Necesitamos conocer: el nombre y apellidos, su correo y su ciudad. Además queremos ver su imagen.  
Crear un botón en html y una capa div. Cuando pulsemos el botón, mostraremos la información del usuario en la capa. Hacerlo con fetch y promesas.

8.- Avanzado. Queremos obtener datos aleatorios de 10 usuarios: <https://randomuser.me/api/?results=10>.  
Mostraremos la foto y su nombre, apellidos, correo, dirección y ciudad.. En html tenemos una capa div. Para cada usuario tendremos que crear un div desde js, y unirlo al div de la página html.  
Cada usuario, tendrá un botón para poder cambiar aleatorioamente **ese usuario**.  
Hacerlo con fetch y promesas.

					
Christina Simmons christina.simmons@example.com Northaven Rd,3807 Dubbo, (New south wales)	Jen Lawson jen.lawson@example.com York Road,5920 Celbridge, (Donegal)	Peremisl Burluk peremisl.burluk@example.com Provulok Dobrolyubova,1215 Snigurivka, (Hmelnicka)	Nayana Kulkarni nayana.kulkarni@example.com Carter Rd Promenade,7738 Saharanpur, (Assam)	Ryder Walker ryder.walker@example.com Ravensbourne Road,7640 Lower hutt, (Gisborne)	مریم گلشن mrym.glsn@example.com بلال جنبی, 928 خرم آباد, (قزوین)
<input type="button" value="Cambiar"/>	<input type="button" value="Cambiar"/>	<input type="button" value="Cambiar"/>	<input type="button" value="Cambiar"/>	<input type="button" value="Cambiar"/>	<input type="button" value="Cambiar"/>
					
Albane Riviere albane.riviere@example.com Rue du Château,7171 Villeurbanne, (Mayenne)	Julia Esteban julia.esteban@example.com Calle de Alcalá,1238 Gijón, (Galicia)	Anatoliy Zhezherin anatoliy.zhezherin@example.com Borshchagivska,9233 Boyarka, (Sumka)	Halvor Kaldestad halvor.kaldestad@example.com Grubbegata,1101 Kragersø, (Rogaland)		
<input type="button" value="Cambiar"/>	<input type="button" value="Cambiar"/>	<input type="button" value="Cambiar"/>	<input type="button" value="Cambiar"/>		

9.- Vamos a crear un blog. La página HTML tiene un título en h1, un botón (Load more) y un buscador.

- Por defecto, **al entrar**, se usará la api: [jsonplaceholder.typicode.com/posts? page=1& limit=5](https://jsonplaceholder.typicode.com/posts?page=1&limit=5) para mostrar 5 posts, con su título y cuerpo. Los posts irán en la capa con id="posts-list" de la página html. Cada post será un div que colgará de la capa con id="posts-list"

- El botón 'Load More', cargará otros nuevos 5 posts, en la misma capa del punto anterior. Pero la llamada a la api varía, y cada vez que pulsemos el botón, cambiaremos a la pagina siguiente, es decir: <https://jsonplaceholder.typicode.com/posts?page=2&limit=5>

**El atributo page varia cada vez que pulsemos el botón.**

- Cuando se pulse sobre cada post, hará otra petición ajax a los comentarios del post. Para ello es importante saber el **id del post**. Api: /posts/id\_del\_post/comments . Los comentarios irán en la capa con id="post-details", debajo del botón Load More. Cada comentario será un div que colgará de la capa con id="post-details".
- El buscador podrá cargar en la capa con id="post-details", el post cuyo numero indiquemos en el cuadro de texto. Si no se escribe ningún número, se avisará con mensaje alert.

**NOTA: Hacer las peticiones ajax con api fetch y await.**

- Código html:

```
<div id="blog-container">
  <h1>Blog</h1>
  <div id="posts-list"></div>
  <button id="load-more-btn">Load More</button>
  <div id="post-details"></div>
  <input type="text" id="search-input" placeholder="Search..." />
  <button id="search-btn">Search</button>
  <div id="loading-spinner" class="spinner"></div>
</div>
```

10.- Avanzado. Vamos a cargar un mapa de la Agencia Estatal de Meteorología. Para ello, en primer lugar tendremos que darnos de alta como desarrolladores y obtener una key:

<https://opendata.aemet.es/centrodedescargas/obtencionAPIKey>

La página HTML tendrá los siguientes elementos:

```
<body>
  <h1>Imagen del día de la AEMET</h1>
  <form>
    <label for="apikey">Pega tu API Key</label><br />
    <textarea name="apikey" id="apikey" cols="30" rows="10"></textarea><br />
    <button id="b1">Cargar mapa</button>
  </form>
  <div id="mapa"></div>
</body>
```

Cuando pulsemos el botón b1, llamaremos a la siguiente api, concatenando la clave recibida.  
**[https://opendata.aemet.es/opendata/api/mapasygraficos/analisis?api\\_key=](https://opendata.aemet.es/opendata/api/mapasygraficos/analisis?api_key=)**

IMP: En el fetch hay que incluir un segundo parámetro para incluir cabeceras:

```
var headers = new Headers({ "cache-control": "no-cache", });
var conf = {
  method: "GET",
  mode: "cors",
  headers: headers,
};
```

La llamada devolverá varios datos, nosotros tendremos que fijarnos en el atributo **datos**. (Recuperar con el método `json()`). Con ese atributo debemos hacer una segunda llamada a la api que nos devolverá el mapa. (Recuperar con el método `blob()`). Una vez obtenido el blob, para pintarlo como atributo `src` de la imagen en js, escribid: `img.setAttribute("src", URL.createObjectURL(mapa));`, donde `mapa` es lo que ha devuelto el método `blob()`.

### Imagen del día de la AEMET

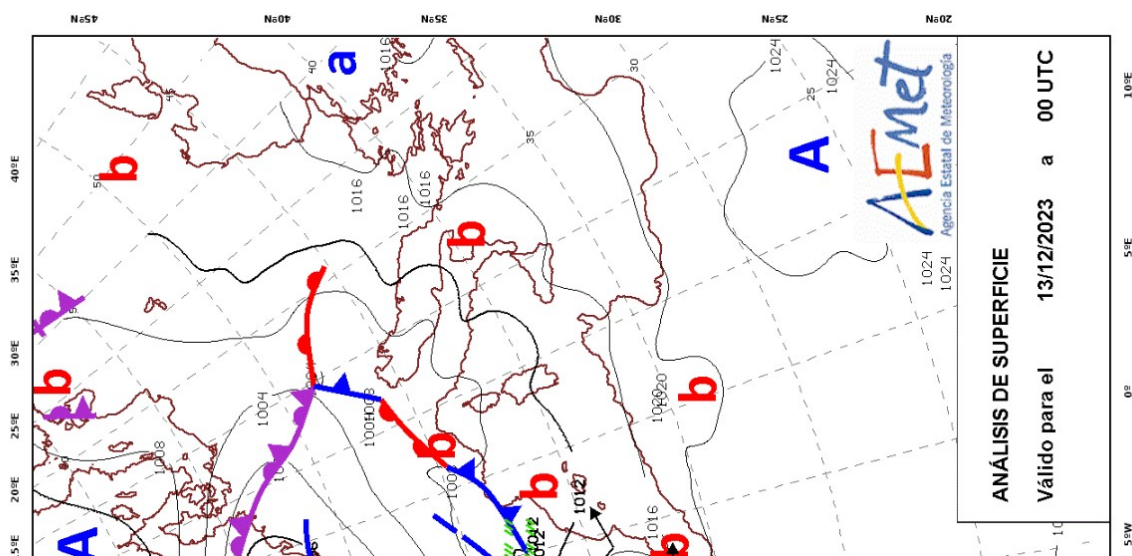
Pega tu API Key

```

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJp
WxlnbRlZG9AZ21hbmVwY29tIiwianRpIjo
iYmVhYXZlZ2NTZ2k1MDMxQ009YmVwY29tIiwiaWF0Ijoi
Mj01MjYxOTg3ZW9hcnRlbiJ9.QMGNVRQ
iLC3pYXQ0je3MDI0NTU4NjMsInVzZXZ3Z
C16fMmFmVWwEwUj5LTAzNDAaNGM2M04NDB
jLWZkOTNmNTk4NzkyYSInJnVzGUiOiIi
Q. Iukc3AJw-
oR5F_c26v6x3MyMFKT5nXhw1Mmxa5x0CIU

```

Cargar mapa



11.- Vamos a conectarnos a un servicio de la NASA para poder ver la imagen de día. Para ello, vamos a registrarnos en: <https://api.nasa.gov/index.html> para obtener una api key.

Nos vamos a conectar a la api: [https://api.nasa.gov/planetary/apod?api\\_key=xxx&date=yyyy-mm-dd](https://api.nasa.gov/planetary/apod?api_key=xxx&date=yyyy-mm-dd)  
Donde la api\_key es el valor obtenido, y la fecha será el día que queremos consultar en formato yyyy-mm-dd.

Para ello, en html tendremos lo siguiente:

```
<body>
  <figure id="imagen"></figure>
  <main>
    <p>Cambiar fecha</p>
    <input type="date" id="fecha" /><br />
  </main>
</body>
```

Cuando se cargue la página, con un prompt pedimos la api key. Y haremos la llamada a la api con la fecha de hoy. La imagen la tendremos que cargar en el id="imagen".

Además, cuando el campo fecha **cambie**, haremos la llamada a la api con el día que haya en el input.

Después de la llamada, tendremos que tomar el atributo url de la respuesta. Es la imagen que tendremos que cargar en el atributo src de la imagen.

Imagen para el día 12/12/2023





## 12.- Avanzado. Animación de usuarios:

Vamos a utilizar la api randomuser.me para mostrar 50 caras aleatorias, cambiando de forma aleatoria 100 veces cada 2 décimas de segundo (200ms). Para ello:

1. Haremos una llamada a la api para que devuelva 1000 imágenes: <https://randomuser.me/api?results=1000>.
2. Construiremos un array y guardaremos las 1000 fotos en tamaño large ( ver respuesta de la api).
3. Cada 200 milisegundos, haremos una mezcla y “barajaremos” el array: para cada elemento del array, intercambiamos la imagen de esa posición y otra calculada aleatoriamente entre 0 y esa posición.
4. De ese array, mostraremos las 50 primeras imágenes en una capa div de html.
5. Parará de mostrar caras aleatorias cuando se haya producido 100 cambios.

El código HTML es:

```
<body>
| <div id="contenido"></div>
</body>
```

## 13.- Vamos a mostrar datos de superhéroes. Dado el siguiente código html:

```
<header id="header"></header>
<section id="section"></section>
```

Nos conectaremos a la api: <https://mdn.github.io/learning-area/javascript/oojs/json/superheroes.json>

Una vez obtenidos los datos, cargaremos en la cabecera header una etiqueta h1 con el nombre del equipo. Y un párrafo <p> con la ciudad y el año en que se formó.

En la sección section, cargaremos los miembros del equipo. Cada uno en un div. Dentro del div, habrá la siguiente información:

- En <h2> : el nombre del superheroe.
- En un <p> la identidad secreta.
- En un <p> la edad.
- En un <p> el titulo: superpoderes
- En una lista ul, la lista de los superpoderes,

Hacer el ejercicio con XMLHttpRequest, fetch con promesas y fetch con await.



## **SUPERHERO SQUAD**

Hometown: Metro City // Formed: 2016

### **MOLTCULL MAN**

Secret identity: Dan Jukes

Age: 29

Superpowers:

- Radiation resistance
- Turning tiny
- Radiation blast

### **MADAME SUPERPUNCH**

Secret identity: Jane Wilson

Age: 39

Superpowers:

- Million tonne punch
- Damage resistance
- Superhuman reflexes

### **ETERNAL JANE**

Secret identity: Unknown

Age: 1000000

Superpowers:

- Immortality
- Heat Immunity
- Inferno
- Teleportation
- Interdimensional travel