

Acceso asíncrono al servidor

Ajax

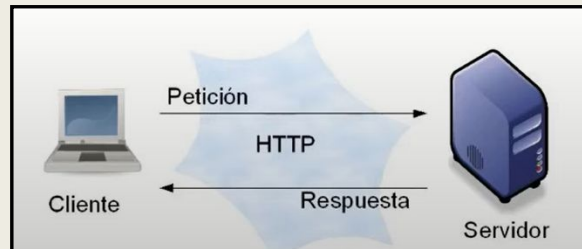
Arquitectura cliente-servidor

Base de cualquier intercambio de datos en la web mediante peticiones a un servidor con el **protocolo HTTP**

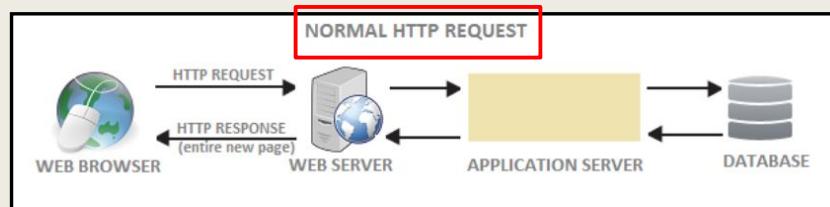
- **cliente:** se refiere a la app web, app móvil, app de escritorio
- **servidor:** se refiere al programa que corre en un servidor y maneja peticiones del cliente, maneja la lógica del sistema, se conecta a la bd, etc...

¿Qué es una petición http?

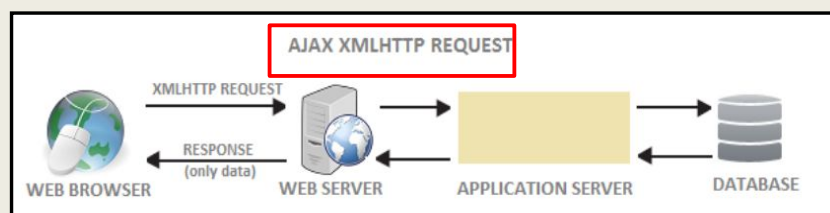
- Acción por parte del navegador de solicitar a un servidor un documento o archivo (*un fichero .html, una imagen, un archivo .js, etc*)



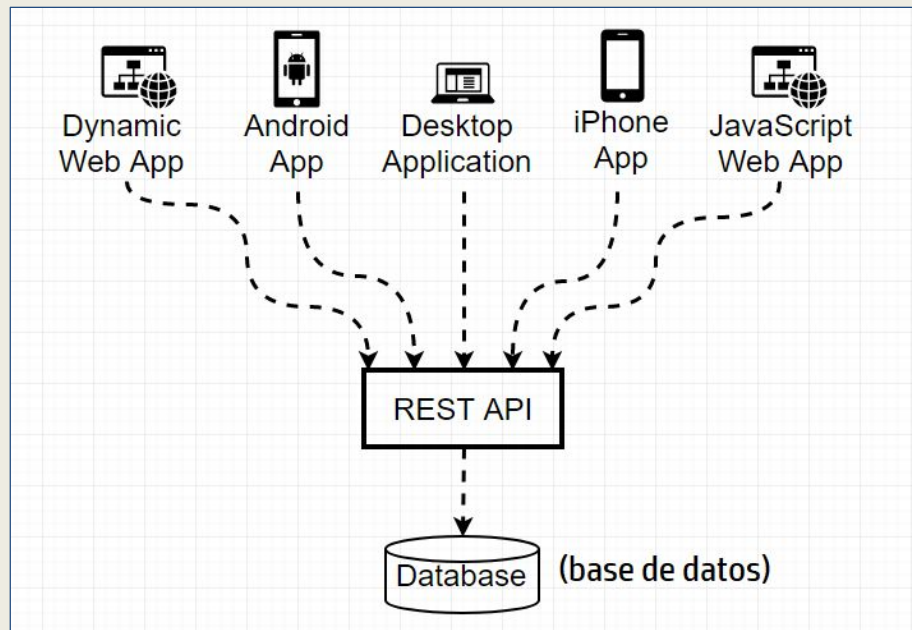
¿Qué es una petición http?



AJAX fue un hito en el desarrollo web y es el concepto central detrás de muchas tecnologías modernas como React.



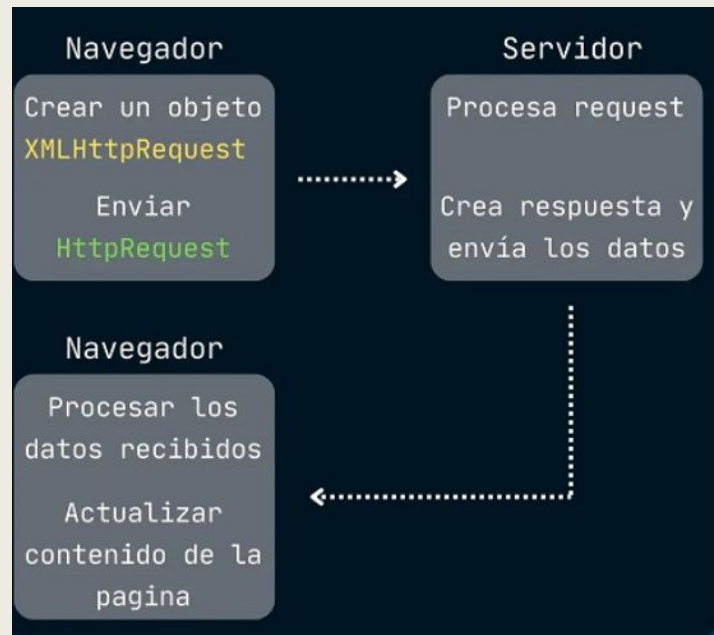
¿Qué es una petición http?



AJAX qué es?








AJAX cómo funciona?



Peticiones asíncronas

- Petición asíncrona: operación que mientras es procesada deja libre al navegador para que pueda hacer otras operaciones.
- Gracias a las llamadas AJAX una página web puede ser **parcialmente actualizada** sin tener que recargar todo el contenido *(lo que conlleva una mejor velocidad de carga y usabilidad)*.

Métodos de peticiones Ajax

Método	Descripción
XMLHttpRequest	Se suele abreviar como XHR . El más antiguo, y también el más verbose. Nativo. 
fetch	Nuevo sistema nativo de peticiones basado en promesas. Sin soporte en IE.
Axios 	Librería basada en promesas para realizar peticiones en Node o navegadores. 
superagent 	Librería para realizar peticiones HTTP tanto en Node como en navegadores.
frisbee 	Librería basada en fetch. Suele usarse junto a React Native.

¿Cómo lo probamos en el aula?

A falta de configurarnos un servidor **consumiremos datos de APIs públicas.**

pero... ¿qué es una API?



APIs - un término muy amplio... - (Interfaces de programación de aplicaciones)

- Una API es “**código**” que dos aplicaciones utilizan para comunicarse.
- Se trata de un conjunto de definiciones y protocolos predefinidos que dan acceso de manera segura y confiable al backend de una aplicación
- Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados, lo que simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero.

APIs - un término muy amplio... - Interfaces de programación de aplicaciones)

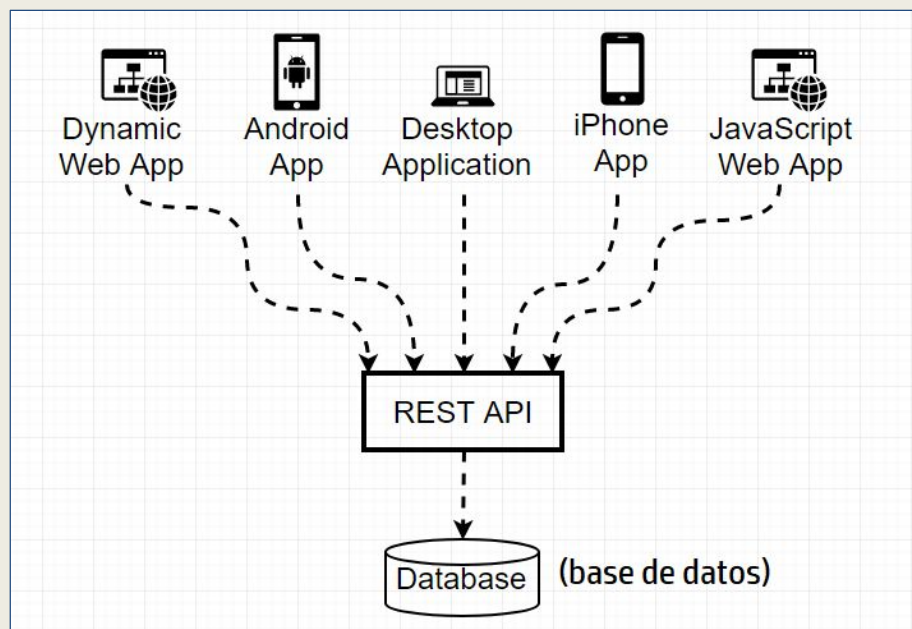
- Normalmente, las APIs van acompañadas de **documentación** que detalla cómo se realiza el intercambio de información.
- Pueden ser **privadas** para el uso de una empresa, abiertas sólo para partners, o **públicas** para que cualquier desarrollador interactúe con ellas. *También pueden ser API **locales** para aplicaciones que se comunican dentro de un mismo ambiente o dispositivo, o **remotas** para cuando hay que acceder a otro punto diferente.*

API - Interfaz de programación de aplicaciones

- Cuando creamos el Front para una aplicación necesitaremos “consumir” datos. Usualmente, estos datos provienen de una API:
 - *nuestro front “consume” estos datos y los presenta o “pinta” en el cliente*
 - *en última instancia, podemos pensar en ellas como “un acceso directo a la base de datos de un servicio web”*
- Proporcionan un conjunto de métodos que el desarrollador puede utilizar a través de HTTP para enviar y recibir datos.

APIs - un término muy amplio... -

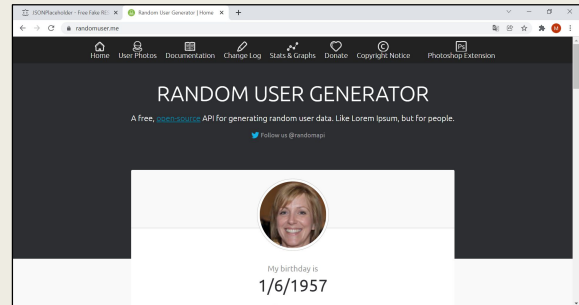
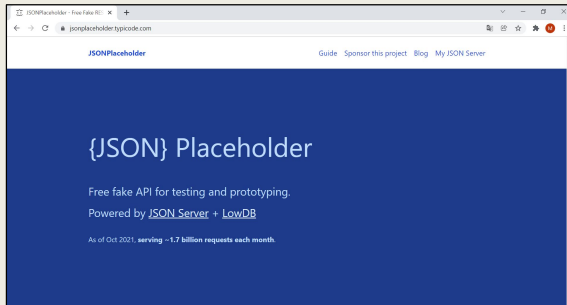
Interfaces de programación de aplicaciones)



APIs - un término muy amplio... - (Interfaces de programación de aplicaciones)






Ejemplos APIs públicas:

<https://jsonplaceholder.typicode.com/> y <https://randomuser.me/>



*La mayor parte de la información que vamos a usar en nuestra página web va a venir en formato **Json**, que una vez recuperada con TypeScript/JavaScript no es más que un objeto o un array que podremos manipular*

Métodos de peticiones Ajax

Método	Descripción
XMLHttpRequest	Se suele abreviar como XHR . El más antiguo, y también el más verbose. Nativo. 
fetch	Nuevo sistema nativo de peticiones basado en promesas. Sin soporte en IE.
Axios 	Librería basada en promesas para realizar peticiones en Node o navegadores. 
superagent 	Librería para realizar peticiones HTTP tanto en Node como en navegadores.
frisbee 	Librería basada en fetch. Suele usarse junto a React Native.

Peticiones asíncronas con Fetch

- **Fetch**: estándar que reemplaza a XMLHttpRequest y nos permite hacer requests http (*peticiones*) para recuperar recursos.
- **¿Cómo se realiza la petición con Fetch?** básicamente se trata de llamar a **fetch** y pasarle como parámetro la URL de la petición:

*// Se realiza la petición (devuelve una **promesa**)*

```
const request = fetch(url);
```

// Si es resuelta, entonces se ejecuta la función propuesta

```
request.then(function(response) { ... });
```

Promesas

- **Promesa** en JavaScript: objeto que gestiona las peticiones asíncronas (*algo que en principio pensamos que se va a cumplir, pero que puede no obtener el resultado esperado...*)
- Contendrá “*en algún momento*” la respuesta a algo que está sucediendo de forma asíncrona; (*más concretamente un objeto de respuesta (response); una respuesta HTTP.*)

Peticiones asíncronas con Fetch

Promesas - estados

- **pending** (*pendiente*) - este estado se asigna automáticamente después de que la promesa se crea, *significa que hasta ahora la promesa no se cumplió ni se rechazó*
- **fulfilled** (*cumplida*) - la promesa ha sido completada con éxito y el resultado que obtenemos es el valor de la operación
- **rejected** (*rechazada*) - la promesa no se ha completado con éxito y el resultado que obtenemos es la razón por la que esto ocurrió (*el error*);

Promesas

● Métodos disponibles de una promesa

Métodos	Descripción
<code>.then(<small>FUNCTION</small> resolve)</code>	Ejecuta la función callback resolve cuando la promesa se cumple.
<code>.catch(<small>FUNCTION</small> reject)</code>	Ejecuta la función callback reject cuando la promesa se rechaza.
<code>.then(<small>FUNCTION</small> resolve, <small>FUNCTION</small> reject)</code>	Método equivalente a las dos anteriores en el mismo <code>.then()</code> .
<code>.finally(<small>FUNCTION</small> end)</code>	Ejecuta la función callback end tanto si se cumple como si se rechaza.

API Fetch de JavaScript

- **Fetch()** devuelve una **promesa** que será **aceptada** (cuando la petición recibe una respuesta) O **rechazada** (cuando por ejemplo hay un fallo de red o por alguna razón no se ha podido completar la petición)

*// Se realiza la petición (devuelve una **promesa**)*

```
const request = fetch(url);
```

// Si es resuelta, entonces se ejecuta la función propuesta

```
request.then(function(response) { ... });
```

API Fetch de JavaScript

- Al método **.then()** se le pasa una **función callback** cuyo parámetro **response** es la respuesta a la petición realizada (objeto respuesta).

```
const request = fetch("url");  
request.then(function(response) { ... });  
-----  
//reescribiendo el código  
fetch("url")  
  .then(function(response) {  
    /** Código que procesa la respuesta **/})
```

API Fetch de JavaScript

- Utilizando **arrow function**:

```
// reescribiendo el código
fetch("url")
  .then(response => response.text())
```

(el método **.text()** devuelve **otra promesa** con el texto plano de la respuesta...)

API Fetch de JavaScript

Invocamos de nuevo a **.then()** para definir qué hacer cuando la promesa es resuelta:

```
// reescribiendo el código
fetch("url")
  .then(response => response.text())
  .then(data => {
    /** Procesar los datos **/
  }));
```

API Fetch de JavaScript

Y por último, si hay un error lo atrapamos con **.catch()**:

```
// Petición HTTP
fetch("url")
  .then(response => response.text())
  .then(data => console.log(data))
  .catch(err => console.error(err.message));
```

API Fetch de JavaScript

Algunos métodos del objeto response: *(la mayoría de ellos para procesar mediante una promesa los datos recibidos y facilitar el trabajo con dichos datos)*

STRING .text()	Devuelve una promesa con el texto plano de la respuesta.
OBJECT .json()	Idem, pero con un objeto json . Equivalente a usar JSON.parse() .
OBJECT .blob()	Idem, pero con un objeto Blob (binary large object).

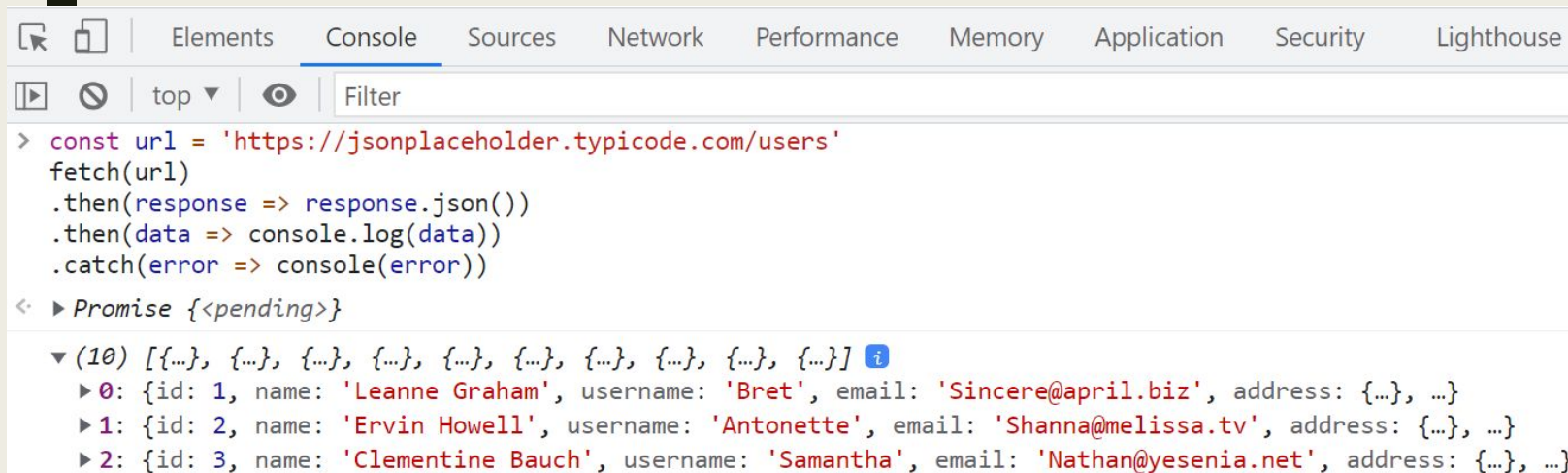
API Fetch de JavaScript

Ejemplo:

```
const url = 'https://jsonplaceholder.typicode.com/users/' //origen de los datos
fetch(url) //solicitud a la url
.then(response => response.json()) //los datos obtenidos se pasan a json
.then(data => console.log(data)) //se lee el objeto data y se muestra
.catch(error => console.log(error))
```

API Fetch de JavaScript

Demo ejecución:



```
> const url = 'https://jsonplaceholder.typicode.com/users'
fetch(url)
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.log(error))

< ▶ Promise {<pending>}

▼ (10) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}] ⓘ
  ▶ 0: {id: 1, name: 'Leanne Graham', username: 'Bret', email: 'Sincere@april.biz', address: {...}, ...}
  ▶ 1: {id: 2, name: 'Ervin Howell', username: 'Antonette', email: 'Shanna@melissa.tv', address: {...}, ...}
  ▶ 2: {id: 3, name: 'Clementine Bauch', username: 'Samantha', email: 'Nathan@yesenia.net', address: {...}, ...}
```

Async/Await

- “**Azúcar sintáctico**” *característica añadida a Javascript para hacer más sencillo el uso de las promesas en nuestro código.*
- Abandona el modelo de **encadenamiento de .then()** para trabajar de forma más tradicional.
- Palabras clave:
 - **async ()** *se coloca delante de la función para definirla como asíncrona. El resto de la función no cambia.*
 - **await** *operador usado para esperar a una Promise. Solo puede ser usado dentro de una async function. Hace que JavaScript espere hasta que la promesa responda y devuelve su resultado.*

async...await

```
const obtenerDatos = async () =>{
  try {
    const respuesta = await fetch('https://jsonplaceholder.typicode.com/users');
    if (!response.ok) throw Error();
    const datos = await response.json();
    // procesar datos
    console.log(datos);
    console.log('Lo último que verás.')
  }
  catch (err) {
    console.error(err);
  }
}

obtenerDatos();
console.log('Lo primero que verás.');
```

Axios - peticiones get

- librería basada en promesas
- muy ligera *(pesa unos 13K; no se nota en absoluto en una página web a diferencia de, por ejemplo jQuery)*
- seguramente es a día de hoy la librería más optimizada para hacer peticiones en menor tiempo *(hablamos de milisegundos..., que cuando la red es lenta o estamos con datos móviles se pueden notar...)*
- para utilizarla lo único que tenemos que hacer es copiar el correspondiente link *(preferiblemente encima de nuestro script, como primera línea de nuestro código)*

Axios - peticiones get

- uso básico:
 - recibe un objeto *(igual que fetch; basta con indicar el método (get) y la url a la que realizamos la petición)*
 - como trabaja con promesas-----> **.then()** y **.catch()** *para definir cómo se comportará en caso de error y en caso de éxito*

```
button.addEventListener('click', () => {  
  axios({  
    method: 'GET',  
    url: 'https://jsonplaceholder.typicode.com/user'  
  }).then(res => {  
    console.log(res.data)  
  }).catch(err => console.log(err))  
})
```


Axios - peticiones get

si queremos recorrer y pintar los datos:

```
button.addEventListener('click', () => {
  axios({
    method: 'GET',
    url: 'https://jsonplaceholder.typicode.com/users'
  }).then(res => {
    const list = document.getElementById('list')
    const fragment = document.createDocumentFragment()
    for (const userInfo of res.data) {
      const listItem = document.createElement('LI')
      listItem.textContent = `${userInfo.id} - ${userInfo.name}`
      fragment.appendChild(listItem)
    }
    list.appendChild(fragment)
  }).catch(err => console.log(err))
})
```

Axios

Get Data

- 1 - Leanne Graham
- 2 - Ervin Howell
- 3 - Clementine Bauch
- 4 - Patricia Lebsack
- 5 - Chelsey Dietrich
- 6 - Mrs. Dennis Schulist
- 7 - Kurtis Weissnat
- 8 - Nicholas Runolfsdottir V
- 9 - Glenna Reichert
- 10 - Clementina DuBuque