

JS

UT07: SOLICITUDES DE RED Y  
CONTROL DE SESIONES

# ÍNDICE

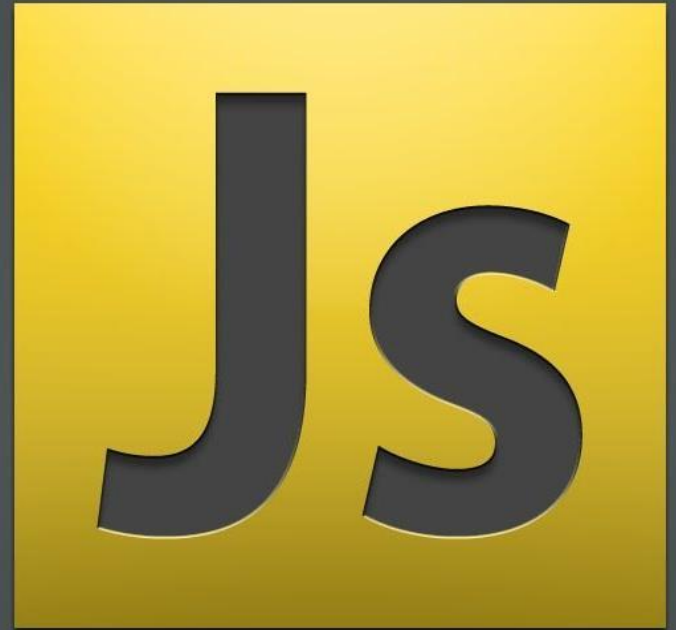
---

- 1.- Introducción a las APIs REST
- 2.- Promesas
- 3.- async/await
- 4.- Promise API
- 5.- Fetch
- 6.- FormData (???)
- 7.- Cookies y document.cookie
- 8.- LocalStorage y SessionStorage
- 9.- IndexedDB (???)
- 10.- Autenticación de usuarios con JWT



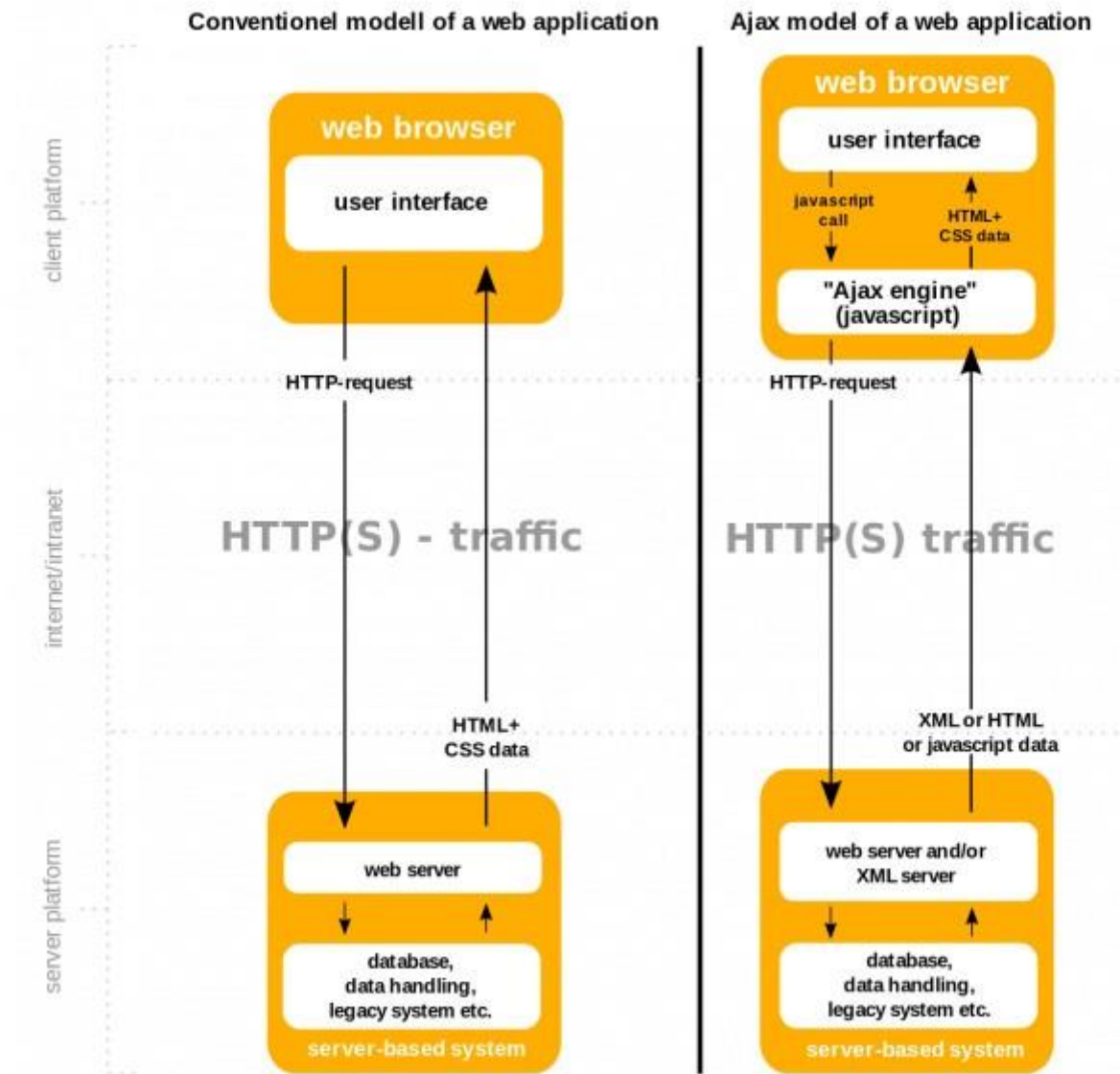
# 1

## INTRODUCCIÓN A LAS APIS REST



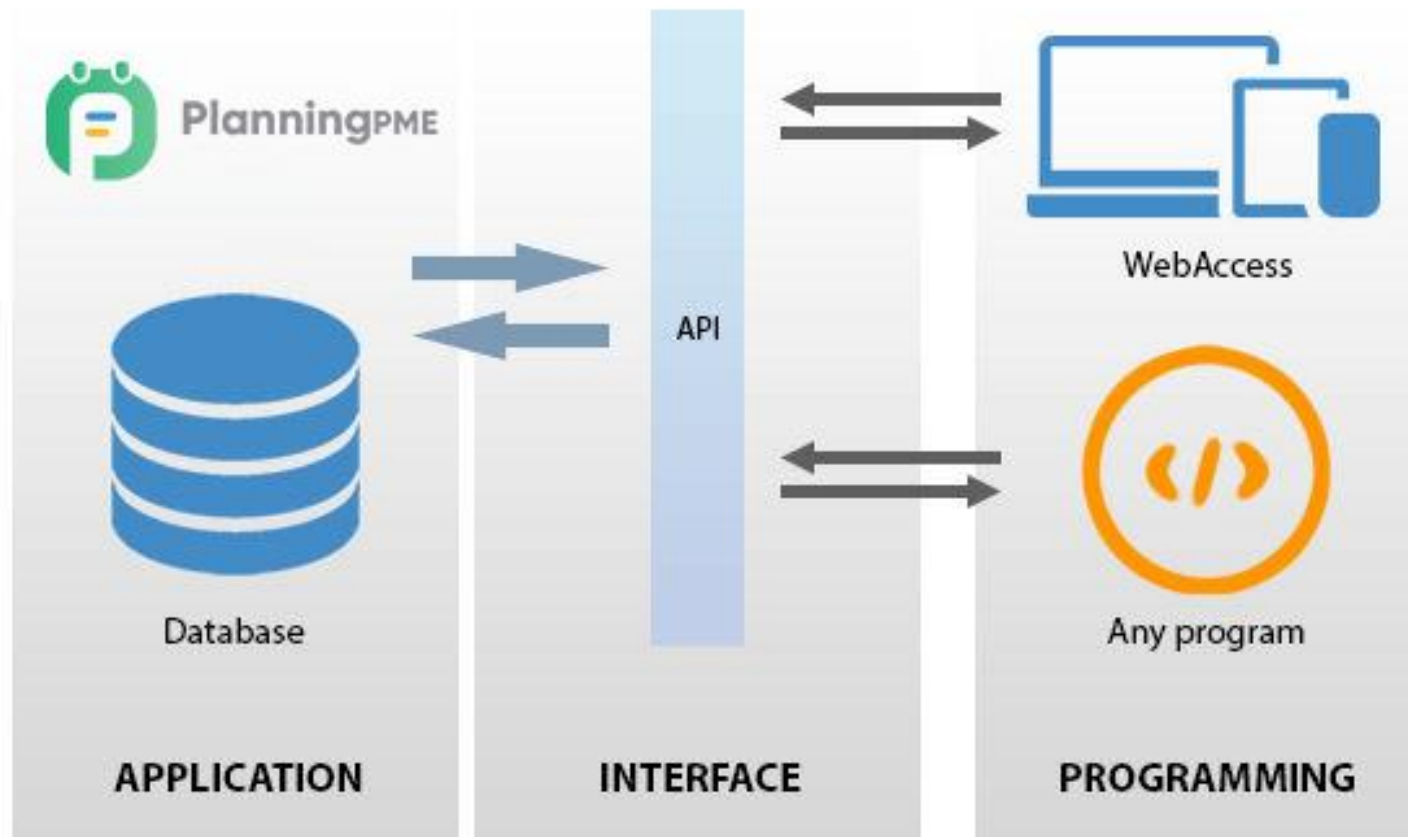
JavaScript permite realizar consultas a un servidor y cargar nueva información en la página.

**AJAX** (Asynchronous JavaScript And XML) es el nombre que se le asigna a las peticiones asíncronas realizadas desde JavaScript (aunque actualmente se usa JSON en lugar de XML).



Aunque las solicitudes desde JavaScript se pueden utilizar para descargar cualquier tipo de recurso, uno de sus principales usos es realizar consultas a APIs REST.

Una **API (Application Programming Interface)** es un conjunto de funcionalidades o recursos que expone un sistema para poder interactuar con él desde otro sistema, independientemente del lenguaje de programación o tecnología de cada uno de ellos.



**REST** (***Representational State Transfer***) es un estilo de arquitectura del software para comunicaciones cliente servidor apoyado en el protocolo HTTP.

REST se basa en tres conceptos clave:

- URLs
- Métodos HTTP
- Estados de respuesta.



## URL (Uniform Resource Locator)

Una URL es la direcci3n que se le da a un recurso en la red. REST redefine este concepto utilizándolo para identificar recursos, pero tambi3n **asignándoles nombres representativos**.

Aś, las consultas a la API son f́cilmente **comprensibles**.

Por ejemplo:

<https://swapi.dev/api/people/1/>

[https://api.twitter.com/2/users/:id/timelines/reverse\\_chronological](https://api.twitter.com/2/users/:id/timelines/reverse_chronological)

## Métodos HTTP

Los métodos HTTP se utilizan para indicar qué se quiere hacer con un recurso determinado.

Se utilizan cuatro métodos principalmente, asociados con las operaciones CRUD:

**GET:** para obtener o leer un recurso.

**PUT:** actualiza o reemplaza un recurso

**DELETE:** elimina un recurso del servidor

**POST:** crea un recurso en el servidor

## Estados de respuesta

El resultado de la consulta a la API se indica en el campo de estado de la respuesta HTTP.

Los estados definidos por el estándar HTTP son:

- 1xx Informational
- 2xx Success
- 3xx Redirection
- 4xx Client Error
- 5xx Server Error

Algunos ejemplos de estados utilizados en REST:

**200 (OK):** la operación solicitada se ha realizado con éxito

**201 (Created):** se ha creado el recurso con éxito en el servidor

**202 (Accepted):** utilizada típicamente para solicitudes que llevan un tiempo para procesar e indica que ha sido aceptada.

**204 (No Content):** usualmente en respuesta a solicitudes PUT, POST y DELETE para indicar que la API REST no devuelve ningún mensaje en el cuerpo del mensaje.

**301 (Moved Permanently):** indica que el modelo de la API ha sido rediseñado y ha cambiado la URI de acceso al recurso.

**307 (Temporary Redirect):** la API REST no procesará la solicitud del cliente. Este tendrá que volver a enviar la solicitud a la URI indicada en el cuerpo de la respuesta. Sin embargo, futuras solicitudes deberán seguir utilizando la URI original.

**400 (Bad Request):** código de error genérico cuando no se adapta ningún otro.

**401 (Unauthorized):** el usuario no ha facilitado el método de autenticación requerido por la API y no tiene acceso al recurso.

**403 (Forbidden):** el usuario no tiene permiso para acceder al recurso.

**404 (Not Found):** indica que la API REST no puede mapear la URI con un recurso, pero puede que sí pueda en un futuro, por lo que sí se permitirían futuras solicitudes.

**405 (Method Not Allowed):** el ḿtodo indicado en la solicitud no est́ permitido para ese recurso, aunque ś lo estarían otros ḿtodos. En la cabecera de la respuesta se suelen incluir los ḿtodos permitidos.

**500 (Server Error):** ćdigo genérico para indicar alǵn tipo de error en el servidor.

**501 (Not Implemented):** el servidor no reconoce la solicitud o el ḿtodo, pero probablemente seŕ una funcionalidad futura.



<https://restfulapi.net/http-status-codes/>

Para realizar consultas REST desde el navegador podemos utilizar el **addon de Firefox RESTClient**.

The image shows two screenshots. The top screenshot is the RESTClient interface, which includes a 'Request' section with a method dropdown set to 'GET' and a URL field containing 'http://www.example.com'. Below this is a 'Body' section with a text area labeled 'Request Body'. The bottom screenshot shows the Firefox Add-ons store page for RESTClient, featuring a star rating of 4.1 stars, 145 reviews, and a description: 'RESTClient, a debugger for RESTful web services. by Chao ZHOU'.

RESTClient, a debugger for RESTful web services.  
by Chao ZHOU

Rating	Count
5 stars	88
4 stars	25
3 stars	10
2 stars	4
1 star	18

Alternativamente, se puede utilizar **Postman**, una aplicación de escritorio con muchas más opciones.

The image shows the Postman website on the left and the Postman desktop application on the right. The website has a navigation bar with links to Product, Pricing, Enterprise, Resources and Support, and Explore. Below the navigation bar is a search bar and a 'Sign In' button, followed by a 'Sign Up for Free' button. The desktop application interface is shown in a window titled 'Postman'. It features a sidebar with a 'Collections' list, including 'Twitter API v2' and 'Single Tweet'. The main panel displays a 'GET' request to 'https://api.twitter.com/2/tweets/{id}'. The 'Params' tab is active, showing a table with 'Key', 'Value', and 'description'. The 'Body' tab is also visible, showing a JSON response. The 'Documentation' panel on the right provides details about the endpoint and its parameters.

Product ▾ Pricing Enterprise ▾ Resources and Support ▾ Explore

Search Postman

Sign In Sign Up for Free

### Debu \_ APIs together

Over 20 million developers use Postman. Get started by signing up or downloading the desktop app.

jsmith@example.com Sign Up for Free

Download the desktop app

Windows Apple Linux

Home Workspaces ▾ API Network ▾ Explore

Twitter API v2 / Tweet Lookup / Single Tweet

GET https://api.twitter.com/2/tweets/{id}

Params Authorization Headers Body Pre-request Scripts Tests Settings Cookies

Query params

Key	Value	description
tweet.fields		Comma-separated list of fields to expand. Expansions e...
expansions		Comma-separated list of fields for the poll object. Exp...
media.fields		Comma-separated list of fields for the place object. Exp...
poll.fields		Comma-separated list of fields for the user object. Exp...
place.fields		
user.fields		

Path Variables

Key	Value	description
id	1403216129661628420	Required. Enter a single Tweet ID.

Body Cookies Headers Test Results

200 OK 488 ms 734 B Save Response ▾

Pretty Raw Preview Visualize JSON ▾

```
1 {
2   "data": {
3     "id": "1403216129661628420",
4     "text": "Doevan Mitchell went down after a collision with Paul George towards the end of
5     game 2. https://t.co/Y93XNDL0n"
```

Documentation

https://api.twitter.com/2/tweets/{id}

This endpoint returns details about the Tweet specified by the requested ID.

For full details, see the [API reference](#) for this endpoint.

Authorization Bearer token

This request is using an authorization helper from collection [Twitter API v2](#)

Request params

tweet.fields Comma-separated list of fields for the Tweet object.

Allowed values: attachments,author\_id,context\_annotations,conversation\_id,created\_at,entities,geo,lang,reply\_to\_user\_id,lang,non\_public\_metrics,organic\_metrics,possibly\_sensitive,pre\_moderated\_metrics,public\_metrics,referenced\_tweets,reply,settings,source,text,withheld

Default values: id,text

OAuth1.0a User Context authorization required if any of the following fields are included in the request: non\_public\_metrics,organic\_metrics,promoted\_metrics

View complete collection documentation ▸

Desktop agent Bootcamp Runner Trash

### What is Postman?

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.







# Ejemplo de consulta a StarWars API

Authentication ▾ Headers ▾ View ▾

Favorites ▾ Data migration RESTClient

[-] Request

Method GET ▾ URL  https://swapi.dev/api/people/1  ▾ 

SEND

Body

Request Body

[-] Response

Headers

Response

Preview

Status Code: 200 OK

allow: GET, HEAD, OPTIONS

content-type: application/json

date: Wed, 28 Dec 2022 07:56:35 GMT

etag: "ee398610435c328f4d0a4e1b0d2f7bbc"

server: nginx/1.16.1

strict-transport-security: max-age=15768000

vary: Accept, Cookie

x-firefox-spdy: h2

x-frame-options: SAMEORIGIN

[-] Response

Headers

Response


Preview

```
1 {
2   "name": "Luke Skywalker",
3   "height": "172",
4   "mass": "77",
5   "hair_color": "blond",
6   "skin_color": "fair",
7   "eye_color": "blue",
8   "birth_year": "1988Y",
9   "gender": "male",
10  "homeworld": "https://swapi.dev/api/planets/1/",
11  "films": ["https://swapi.dev/api/films/1/", "https://swapi.dev/api/films/2/", "https://swapi.dev/api/films/3/", "https://swapi.dev/api/films/6/"],
12  "species": [],
13  "vehicles": ["https://swapi.dev/api/vehicles/14/", "https://swapi.dev/api/vehicles/30/"],
14  "starships": ["https://swapi.dev/api/starships/12/", "https://swapi.dev/api/starships/22/"],
15  "created": "2014-12-09T13:50:51.644000Z",
16  "edited": "2014-12-20T21:17:56.891000Z",
17  "url": "https://swapi.dev/api/people/1/"
```

[-] Curl1



Command

```
curl -X GET -k -i 'https://swapi.dev/api/people/1'
```

 Authentication ▾ Headers ▾ View ▾

Favorites ▾ Data migration RESTClient

[-] Request

Method GET ▾ URL  https://swapi.dev/api/people/1251  ▾ 

SEND

Body

Request Body

[-] Response

Headers



Response

Preview

Status Code	: 404 Not Found
allow	: GET, HEAD, OPTIONS
content-type	: application/json
date	: Wed, 28 Dec 2022 07:58:30 GMT
etag	: "8bee5c3ad44d6c57f19e49e8e76ee09e"
server	: nginx/1.16.1
vary	: Accept, Cookie
x-firefox-spdy	: h2
x-frame-options	: SAMEORIGIN

[-] Curl

Command

```
curl -X GET -k -i 'https://swapi.dev/api/people/1251'
```

## Algunas APIs permiten pasar parámetros en la URL

**Searching**

All resources support a `search` parameter that filters the set of resources returned. This allows you to make queries like:

```
https://swapi.dev/api/people/?search=r2
```

All searches will use case-insensitive partial matches on the set of search fields. To see the set of search fields for each resource,

**[ - ] Request**

Method  URL

**Body**

Request Body

**[ - ] Response**

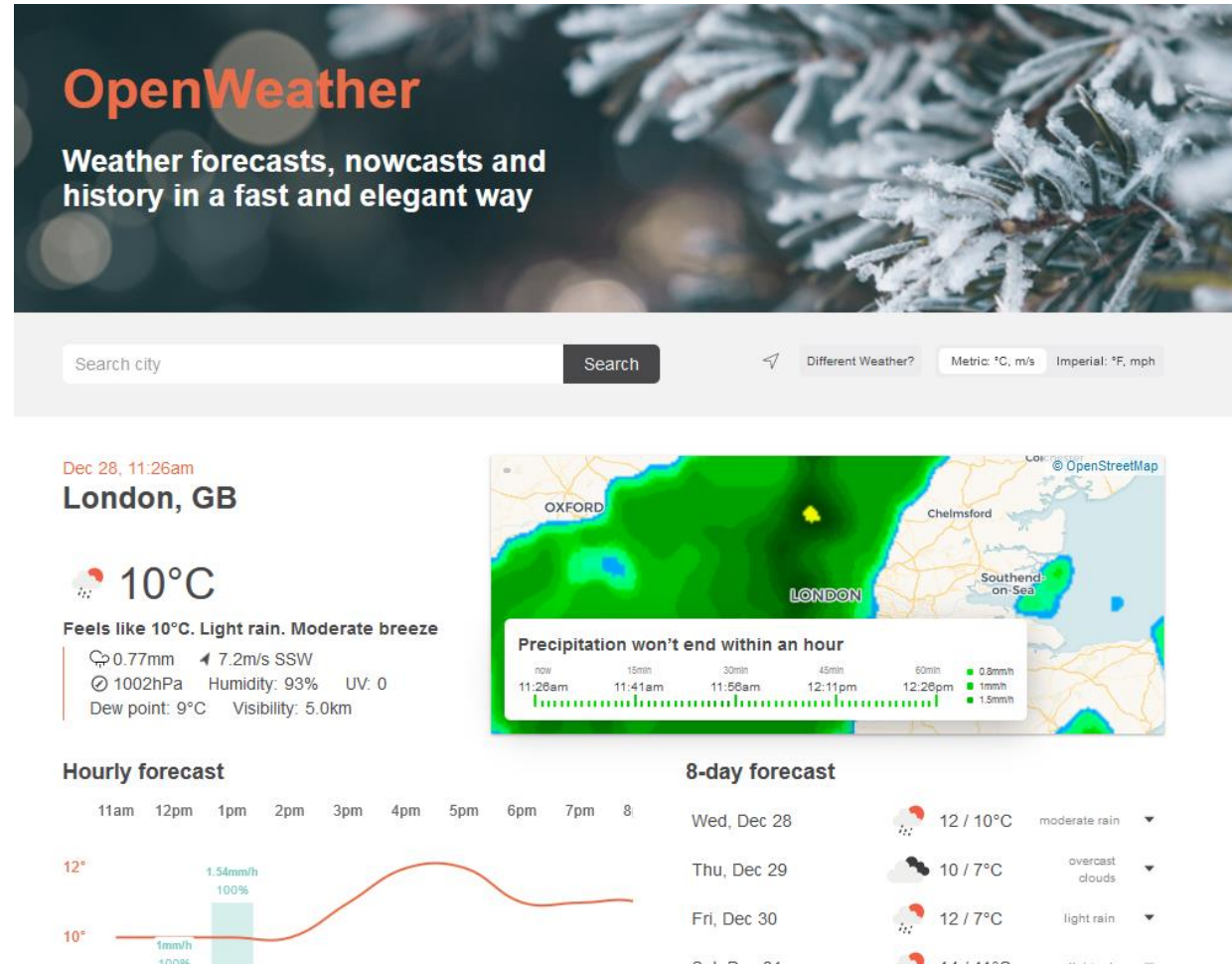
Headers Response Preview

```
1 {
2   "count": 1,
3   "next": null,
4   "previous": null,
5   "results": [{
```

La mayoría de las APIs requieren algún tipo de autenticación, usualmente mediante una **API Key**.

Ejemplo:

**OpenWeather**



## Primero creamos una cuenta

### Create New Account

We will use information you provided for management and administration purposes, and for keeping you informed by mail, telephone, email and SMS of other products and services from us and our partners. You can proactively manage your preferences or opt-out of communications with us at any time using [Privacy Centre](#). You have the right to access your data held by us or to request your data to be deleted. For full details please see the OpenWeather [Privacy Policy](#).

☒ I am 16 years old and over


☐ I agree with [Privacy Policy](#), [Terms and conditions of sale](#) and [Websites terms and conditions of use](#)


I consent to receive communications from OpenWeather Group of Companies and their partners:

☐ System news (API usage alert, system update, temporary system shutdown, etc)

☐ Product news (change to price, new product features, etc)

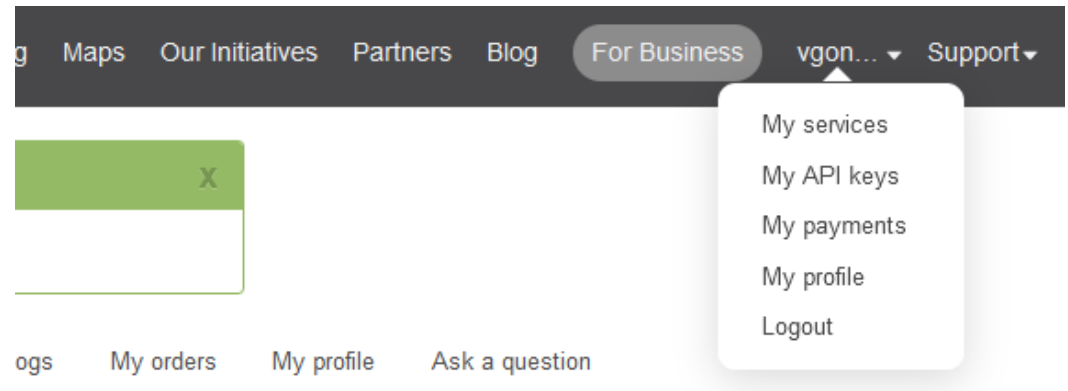
☐ Corporate news (our life, the launch of a new service, etc)

 No soy un robot

  
reCAPTCHA  
[Privacidad](#) - [Términos](#)

Create Account



Y ya podremos acceder a la sección **My API Keys**



En el caso de esta API puedes tener diversas API Keys las cuales se gestionarán desde aquí.

[New Products](#) [Services](#) [API keys](#) [Billing plans](#) [Payments](#) [Block logs](#) [My orders](#) [My profile](#) [Ask a question](#)

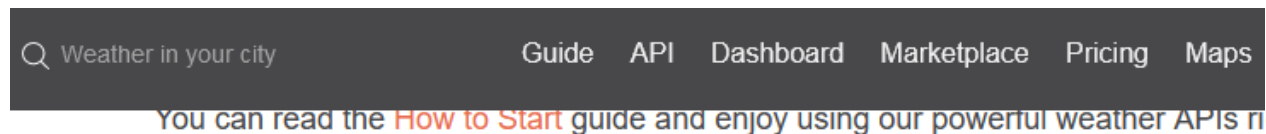
You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.

Key	Name	Status	Actions
ae6ba5815beaa9[REDACTED]	Default	Active	 

Create key

Generate

Falta saber cómo hay que enviar la API Key en las consultas, por lo que acudimos a la documentación.



## Current & Forecast weather data collection

### Current Weather Data

[API doc](#)[Subscribe](#)

- Access current weather data for any location including over 200,000 cities
- We collect and process weather data from different sources such as global and local weather models, satellites, radars and a vast network of weather stations
- JSON, XML, and HTML formats
- Included in both free and paid subscriptions

### Hourly Forecast 4 days

[API doc](#)[Subscribe](#)

- Hourly forecast is available for 4 days
- Forecast weather data for 96 timestamps
- JSON and XML formats
- Included in the Developer, Professional and Enterprise subscription plans

En el primer ejemplo ya podemos ver que la API Key simplemente se pasa como parámetro en la URI

#### API call

---

```
https://api.openweathermap.org/data/2.5/weather?lat={lat}&  
lon={lon}&appid={API key}
```





Vamos a probarla. Necesitamos indicar las coordenadas geográficas de la ubicación a consultar, por ahora las buscamos por internet, pero podríamos obtenerlas con otra consulta a **Geocoding API** tal como se indica en la documentación.

#### Parameters

lat, lon

required Geographical coordinates (latitude, longitude). If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our **Geocoding API**.

#### Coordenadas de León (España)

Aquí podrás obtener las coordenadas geográficas de León, España, de mane grados decimales para que puedas localidar León, España, en Google Maps.

Coordenadas geográficas de León, España, en grados decimales:

- Longitud: -5.5703200
- Latitud: 42.6000300


[-] Request

Method

GET

▼

URL

 https://api.openweathermap.org/data/2.5/weather?lat=42.60&lon=-5.57&appid=ae6ba5815[REDACTED]

Body

Request Body

[-] Response

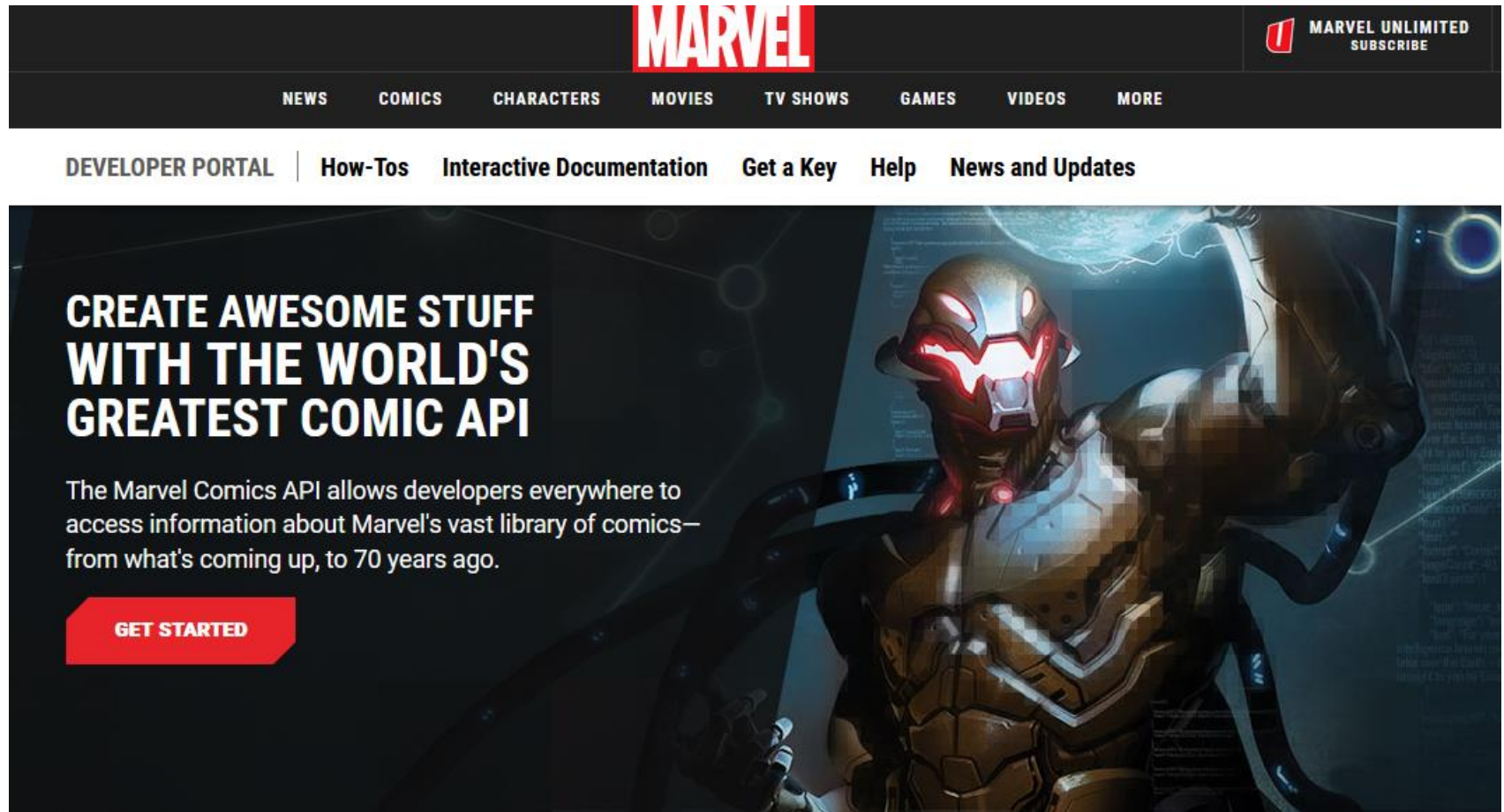
Headers

Response

Preview

```
1 {  
2   "coord": {  
3     "lon": -5.5667,  
4     "lat": 42.6  
5   },  
6   "weather": [{  
7     "id": 804,  
8     "main": "Clouds",  
9     "description": "overcast clouds",  
10    "icon": "04d"  
11  }  
12 }
```

## Otro ejemplo de API: Marvel API



The image shows the Marvel Developer Portal homepage. At the top, there is a dark navigation bar with the Marvel logo in the center. To the right of the logo is a red button that says "MARVEL UNLIMITED SUBSCRIBE". Below the navigation bar is a horizontal menu with links: NEWS, COMICS, CHARACTERS, MOVIES, TV SHOWS, GAMES, VIDEOS, and MORE. Below this menu is a white bar with the text "DEVELOPER PORTAL" followed by a vertical line and then "How-Tos", "Interactive Documentation", "Get a Key", "Help", and "News and Updates". The main content area has a dark background with a large image of Iron Man on the right. On the left, there is white text that reads "CREATE AWESOME STUFF WITH THE WORLD'S GREATEST COMIC API". Below this text is a paragraph: "The Marvel Comics API allows developers everywhere to access information about Marvel's vast library of comics—from what's coming up, to 70 years ago." At the bottom left of this section is a red button that says "GET STARTED".

**MARVEL**

MARVEL UNLIMITED  
SUBSCRIBE

NEWS COMICS CHARACTERS MOVIES TV SHOWS GAMES VIDEOS MORE

DEVELOPER PORTAL | How-Tos Interactive Documentation Get a Key Help News and Updates

**CREATE AWESOME STUFF  
WITH THE WORLD'S  
GREATEST COMIC API**

The Marvel Comics API allows developers everywhere to access information about Marvel's vast library of comics—from what's coming up, to 70 years ago.

**GET STARTED**

Creamos una cuenta y obtendremos acceso a una clave pública y una privada



**SIGN IN**

Username or Email Address

Password

**SIGN IN**

[Need help signing in?](#)

**CREATE AN ACCOUNT**

## MY DEVELOPER ACCOUNT

Hi vgonza31076583!

Here's your personal Marvel Comics API information:

### Your public key

91f5c853261777b74d9f7a884470b760

### Your private key

[Read more about how to use your keys to sign requests. »](#)

**Your rate limit:**

3000  
calls/day

Number of calls your application can make per day.

### Your authorized referrers

List any domains that can make calls to the Marvel Comics API using your API key here:



delete

add a new referrer

Note: List the domain and path only - don't include "http" or other scheme designations. Only use the characters a - z ,

0 - 9, ., -, and \*.

Read more about how to authorize referring domains in browser-based apps and web sites. »

La autenticación depende de si nuestra aplicación está en el cliente o el servidor.

### Authentication for Client-Side Applications

Requests from client-side (browser-based) applications must originate from a pre-authorized web site or browser extension URL. You may add or edit your authorized domains in your API account panel. You may use the "\*" wildcard to denote subdomains or paths. For example:

**marvel.com** - will authorize requests from Marvel.com but no subdomains of Marvel.com

**developer.marvel.com** - will authorize requests from developer.marvel.com

**\*.marvel.com** - will authorize requests from any Marvel.com subdomain as well as Marvel.com

**\*.marvel.com/apigateway** - will authorize requests from the apigateway path on any Marvel.com subdomain as well as Marvel.com

### Authentication for Server-Side Applications

Server-side applications must pass two parameters in addition to the apikey parameter:

**ts** - a timestamp (or other long string which can change on a request-by-request basis)

**hash** - a md5 digest of the ts parameter, your private key and your public key (e.g. md5(ts+privateKey+publicKey))

For example, a user with a public key of "1234" and a private key of "abcd" could construct a valid call as follows:

`http://gateway.marvel.com/v1/public/comics?ts=1&apikey=1234&hash=ffd275c5130566a2916217b101f26150` (the hash value is the md5 digest of 1abcd1234)

En este caso las pruebas con RestClient no funcionarán, ya que lo interpreta como una aplicación del lado del servidor.

[-] Request

Method

GET

▼

URL

 http://gateway.marvel.com/v1/public/comics?apikey=91f5c853261777b74d9f7a884470b760

Body

Request Body

[-] Response

Headers

Response

Preview

1

```
{"code": "MissingParameter", "message": "You must provide a hash."}
```

Podŕamos generar el hash como se indica en la documentaci3n.

For example, a user with a public key of "1234" and a private key of "abcd" could construct a valid call as follows:

`http://gateway.marvel.com/v1/public/comics?ts=1&apikey=1234&hash=ffd275c5130566a2916217b101f26150` (the hash value is the md5 digest of 1abcd1234)

Uso la web <https://www.md5.cz/>

function md5()

Online generator [md5 hash](#) of a string

md5 ( '70c4902f[REDACTED]4d9f7a884470b760 )

hash darling, hash!

You are awesome! Here is your MD5 checksum:

[REDACTED]b780f9c0950a



## [-] Request

Method GET

URL

<http://gateway.marvel.com/v1/public/comics?ts=1&apikey=9f15c695a1b0760&hash=edb4e2c193ff9550a>

## Body

Request Body

## [-] Response

Headers

Response

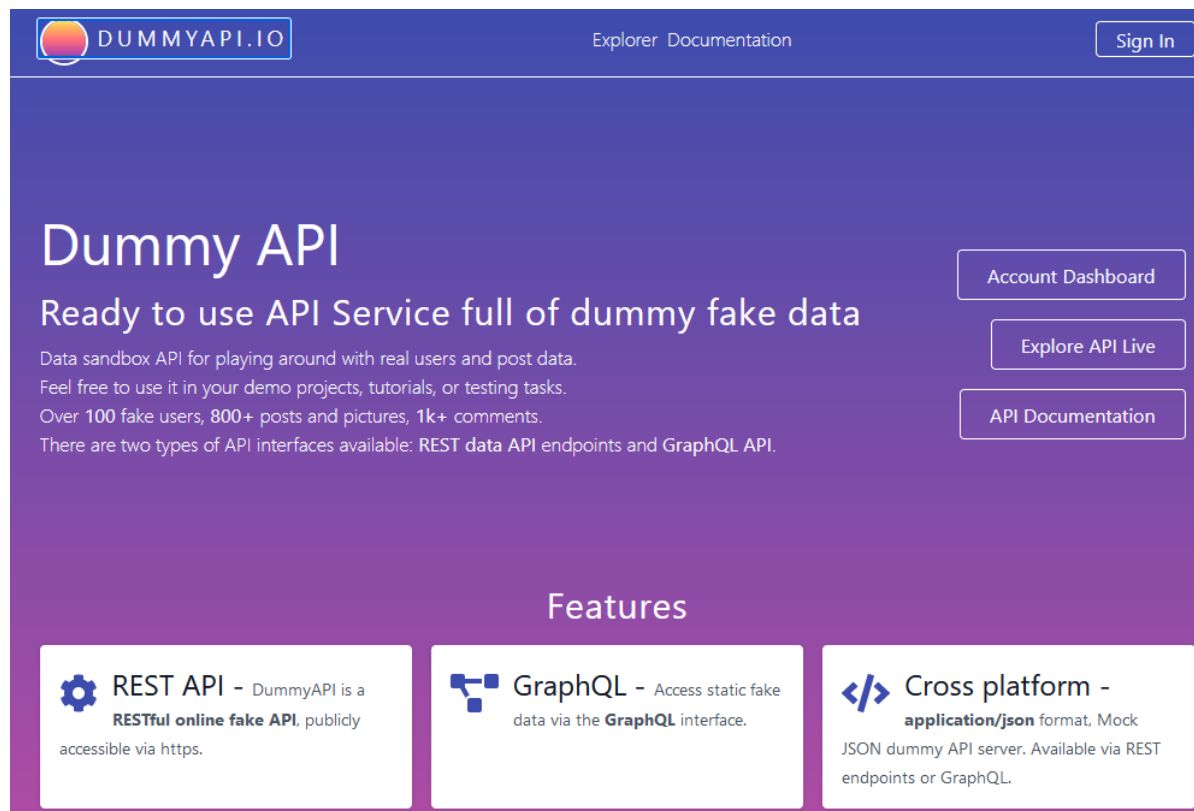
Preview

```
1 {  
2   "code": 200,  
3   "status": "Ok",  
4   "copyright": "© 2022 MARVEL",  
5   "attributionText": "Data provided by Marvel. © 2022 MARVEL",  
6   "attributionHTML": "<a href='\"http://marvel.com\">Data provided by Marvel. © 2022 MARVEL</a>",  
7   "etag": "aa275b9eab52a5c839c2115837c34d3a77405c43",  
8   "data": {  
9     "offset": 0,  
10    "limit": 20,  
11    "total": 53775,
```



En algunas APIs la autenticación se realiza mediante las cabeceras HTTP.

## Ejemplo: Dummy API



## Generamos un API Id



### Account

#### User App ID

Here you can generate personal **APP token** for API.

- ✓ It is necessary to calculate API usage statistic to avoid automatic scrapers.
- ✓ Define a personal environment for each user. Where you can make CRUD operation on entities(user/post/comments etc.). This changes will be visible only for you.
- ✓ Use App ID value to set **app-id** header for all request to API.

#### App IDs list

Dec 29 2022 09:04:04	
63ad49f41cc3[REDACTED]1	

Generate App ID

#### Sponsor Account Setup

Free account is limited to 500 calls per day.

In case you want to exceed this limit, you can became our **patron** on **patreon**. We have a cheapest possible tire **1\$** per month, so you can decide a price by yourself.

BECOME A PATRON

UDP! Free to use with no limits. While new API version is in beta testing!

Si analizamos la documentación veremos que se debe indicar en cada consulta mediante la **cabecera app-id**.

## Headers

It is required to set **app-id** Header for each request.

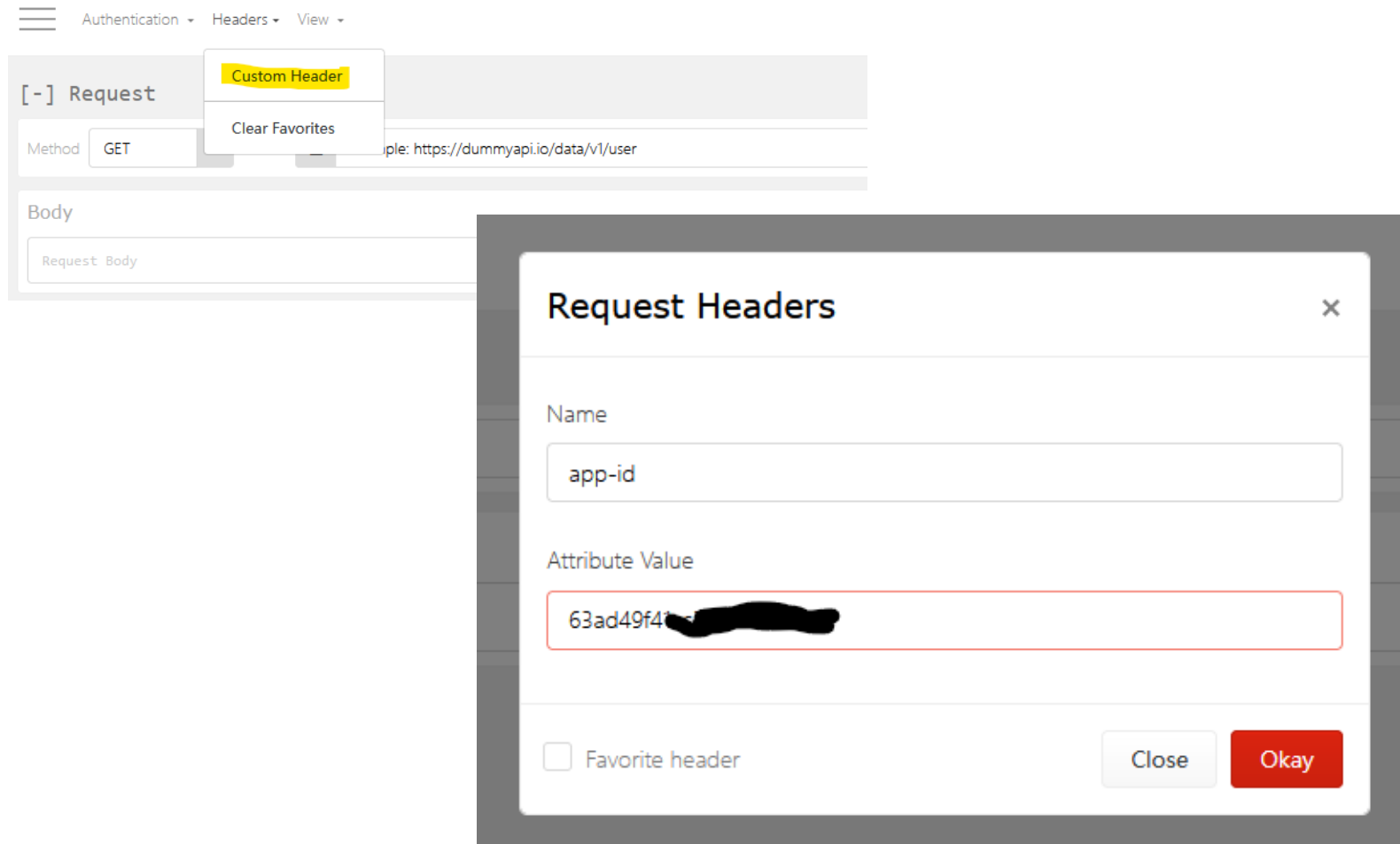
It helps us to determine your personal environment. So only you can access data that were created or update.

You can get personal App ID value on your account page.

You can have as much App ID as you want and use it in parallel(for different projects, envs etc).

**Example:** app-id: 0JyYiOQXQQr5H9OEn21312

Creamos la cabecera en *Headers* -> *Custom header*



## [-] Request

Method

GET



URL

<https://dummyapi.io/data/v1/user>

## Headers

app-id: 63ad49



## Body

Request Body

## [-] Response

Headers

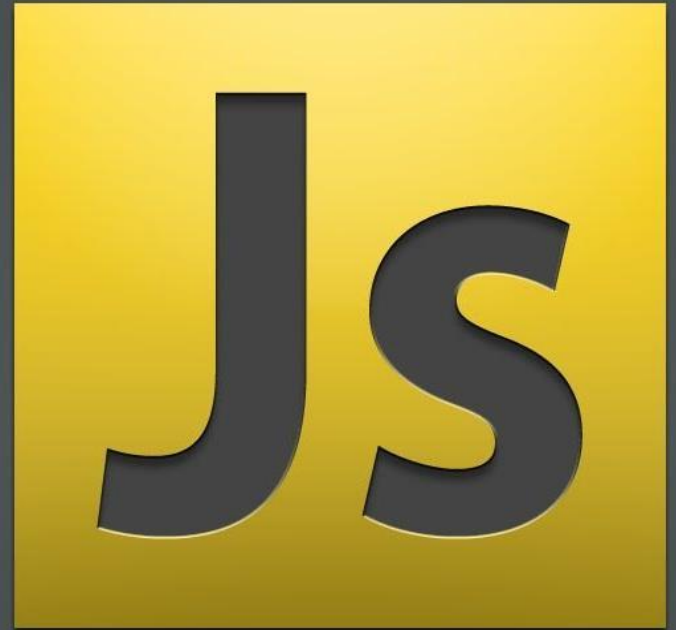
Response

Preview

```
1 {  
2   "data": [{  
3     "id": "60d0fe4f5311236168a109ca",  
4     "title": "ms",  
5     "firstName": "Sara",  
6     "lastName": "Andersen",  
7     "picture": "https://randomuser.me/api/portraits/women/58.jpg"
```

# 2

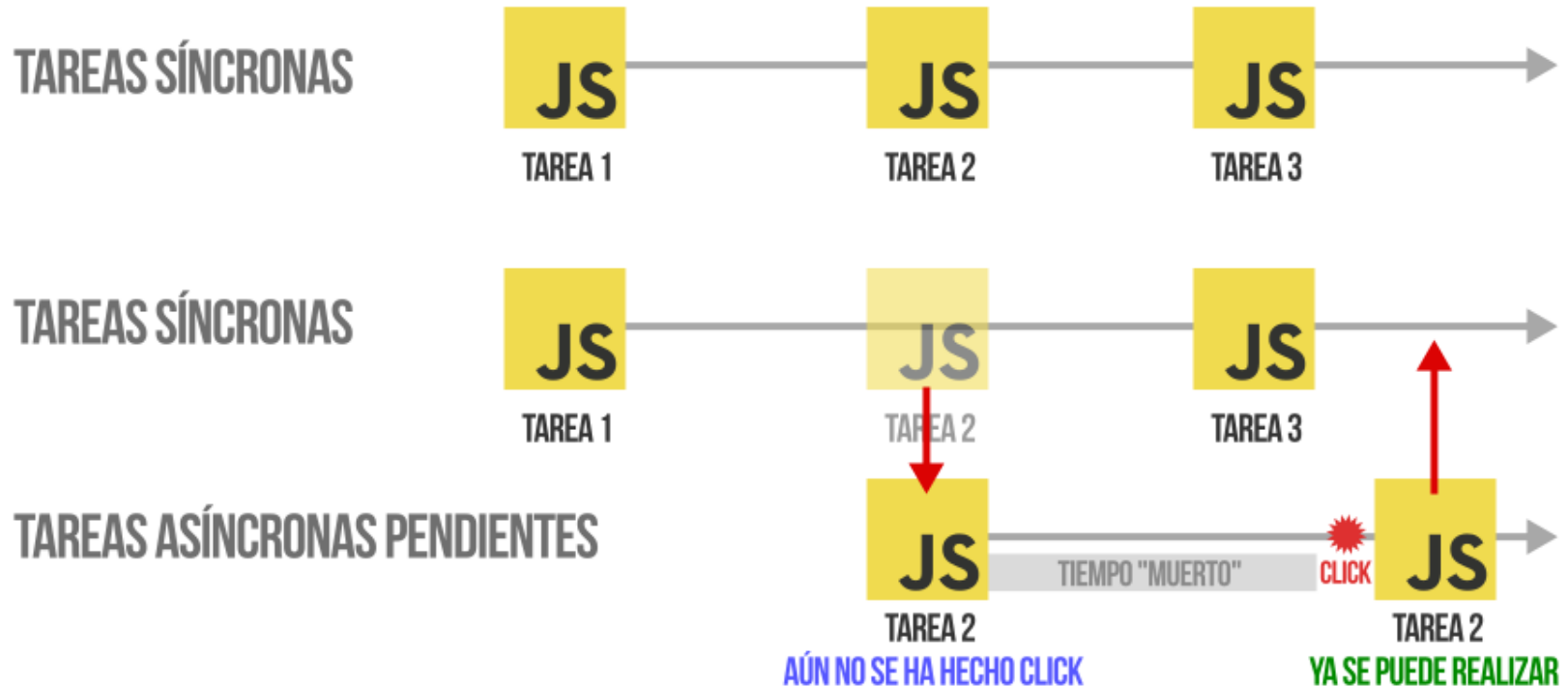
## PROMESAS



Es muy frecuente que las páginas web realicen operaciones asíncronas durante su funcionamiento.

Una **acción asíncrona** es una tarea que no se realiza inmediatamente, sino que se puede demorar un poco (por ejemplo, la carga de un fichero o script).

El programa no espera a que finalice, sino que sigue ejecutándose y ya acabará cuando sea.





**Ejemplo:** carga de un script de forma dinámica desde JavaScript

```
function loadScript(src) {  
    let script = document.createElement('script');  
    script.src = src;  
    document.head.append(script);  
}  
  
loadScript('/my/script.js');  
console.log("Esta línea se ejecuta sin esperar a que cargue");
```

Si en el ejemplo anterior llamara a una función que está en el script descargado, obtendría un error, ya que el navegador no habrá tenido tiempo para descargar el script.

```
function loadScript(src) {  
    let script = document.createElement('script');  
    script.src = src;  
    document.head.append(script);  
}  
  
loadScript('/my/script.js');  
myFunc();    // Esta función está en el script. ERROR
```

¿Cómo hacemos entonces si queremos ejecutar un código que dependa de que haya finalizado una acción asíncrona? La solución son los **callbacks**.

Un **callback** es una función, generalmente anónima, que se pasa como argumento a la función asíncrona para que se ejecute cuando haya finalizado la acción.

Utilizando *callbacks*, el ejemplo anterior quedaría así

```
function loadScript(src, callback) {  
    let script = document.createElement('script');  
    script.src = src;  
    script.onload = () => callback(script);  
    document.head.append(script);  
}  
  
loadScript('https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js', script => {  
    alert(`Genial, el script ${script.src} está  
cargado`);  
    alert( _ ); // _ es una función declarada en el  
script cargado  
});
```

Es muy común cargar elementos secuencialmente, para ello habría que poner un callback dentro del otro.

```
function loadScript(src, callback) {  
    let script = document.createElement('script');  
    script.src = src;  
    script.onload = () => callback();  
    document.head.append(script);  
}  
  
// Solución: ponemos la segunda llamada dentro del callback  
loadScript( urlScript1, function() {  
    console.log('1er callback. Cargamos el segundo script')  
    loadScript( urlScript2, function() {  
        console.log(2º callback);  
    } )  
} )
```

En los ejemplos anteriores no se han tenido en cuenta los errores, ¿qué pasa si el script da un error al cargar?

Esto se puede gestionar **enviando un parámetro de error a la función de *callback***.

Por norma general, el primer parámetro de la función se reserva para el error y el segundo para enviar los datos en caso de resultado exitoso. A esto se le llama **callback error primero**.

```
function loadScript(src, callback) {  
    let script = document.createElement('script');  
    script.src = src;  
  
    // El evento error se dispara si hay algún error al cargar el  
    recurso  
    script.onload = () => callback( null, script );  
    script.onerror = () => callback( new Error(`Error: ${src}`)  
);  
  
    document.head.append(script);  
}  
  
// Solución: ponemos la segunda llamada dentro del callback  
loadScript( urlScript1, function( error, script ) {  
    if (error) {  
        console.log('Aquí manejaríamos el error del script 1');  
    } else {  
        console.log('El script 1 se ha cargado con éxito')  
    }  
} );
```

Aunque los callbacks son útiles cuando hay una o dos acciones asíncronas, si necesitamos ejecutar múltiples acciones asíncronas de forma secuencial el código se puede complicar mucho.

Esto es lo que se llama ***callback hell*** o **pirámide infernal**.



```
loadScript( urlScript1, function( error, script ) {  
  if (error) {  
    console.log('Manejo de error del script1');  
  } else {  
    console.log('El script 1 se ha cargado con éxito')  
    loadScript( urlScript2, function( error, script ) {  
      if (error) {  
        console.log("Manejo de error del script 2");  
      } else {  
        console.log("El script 2 se ha cargado con éxito");  
        loadScript( urlScript3, function( error, script ) {  
          if (error) {  
            console.log("Manejo de error del script 3");  
          } else {  
            console.log("El script 3 se ha cargado con éxito");  
            loadScript( urlScript4, function( error, script ) {  
              if (error) {  
                console.log("Manejo de error del script 4");  
              } else {  
                console.log("El script 4 se ha cargado");  
              }  
            })  
          }  
        })  
      }  
    })  
  }  
});
```

La solución para evitar todos los problemas derivados del uso de *callbacks* son las **promesas**, un mecanismo de JavaScript introducido en ES6 (2015) que es específico para gestionar eventos asíncronos.

La forma de declarar una promesa es la siguiente:

```
let promise = new Promise( function(resolve, reject) {  
                                // El código del ejecutor  
                            });
```

Como se puede ver, una promesa tiene tres partes:

- **Ejecutor:** es el código asíncrono que se va a ejecutar. En el ejemplo anterior con *callbacks* sería la carga del script.
- **resolve:** es la función que se pasa automáticamente al ejecutar y a la que hay que invocar desde dentro de este código cuando el evento asíncrono haya concluido con éxito.
- **reject:** de forma análoga a *resolve*, esta función debe ser invocada si hay un error con el evento asíncrono.

Una vez creada la promesa, el código se seguirá ejecutando.

Para indicar qué hay que hacer cuando la promesa se haya cumplido hay que utilizar la función **then()**.

De forma análoga se puede indicar el código a ejecutar cuando una promesa ha finalizado con error mediante la función **catch()**.

El código anterior quedaría de la siguiente forma si utilizáramos promesas.

```
const promise = new Promise( function(resolve, reject) {  
  // Este es el código que se ejecuta de forma asíncrona  
  let script = document.createElement('script');  
  script.src = url;  
  document.head.append(script);  
  
  // Cuando se ha ejecutado el código se invoca la función resolve()  
  // para indicar que la promesa se ha cumplido  
  script.onload = () => resolve();  
} );  
  
// Con el then indicamos qué hay que hacer una vez que se haya cumplido  
// la promesa  
promise.then( () => {  
  console.log('Se ejecuta una vez que la promesa se haya cumplido.');
```

Las promesas también se pueden **encadenar**. Esto se debe a que la función **then()** devuelve una promesa por lo que puede volver a capturarse con otro *then*.

## Promise.prototype.then()

The `then()` method of `Promise` instances takes up to two arguments: callback functions for the fulfilled and rejected cases of the `Promise`. It immediately returns an equivalent `Promise` object, allowing you to [chain](#) calls to other promise methods.

En el siguiente código podemos ver varios manejadores then que se van ejecutando secuencialmente.

```
new Promise( function(resolve, reject) {  
    setTimeout( () => resolve(1), 1000);  
})  
  .then( function(result) {  
    alert(result);      // 1  
    return result*2;  
  })  
  .then( function(result) {  
    alert(result);      // 2  
    return result*2;  
  })  
  .then( function(result) {  
    alert(result);      // 4  
    return result*2;  
  });
```

Un error muy frecuente es asociar varios manejadores en lugar de encadenarlos, lo que puede que no produzca los resultados esperados.

```
let promise = new Promise( function(resolve, reject) {  
    setTimeout( () => resolve(1), 1000);  
})  
promise.then( function(result) {  
    alert(result);        // 1  
    return result*2;  
});  
promise.then( function(result) {  
    alert(result);        // 1  
    return result*2;  
});  
promise.then( function(result) {  
    alert(result);        // 1  
    return result*2;  
});
```



Cuando una promesa no se cumple, se puede capturar con la función **catch()**

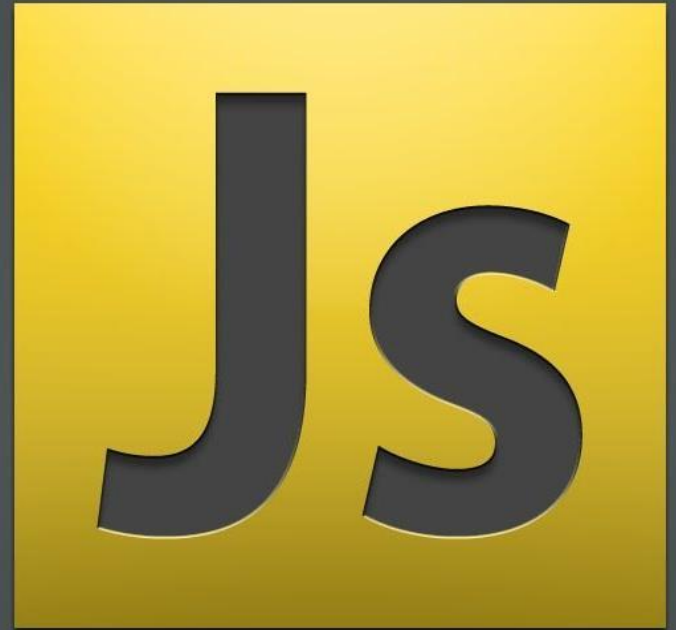
```
let promise = new Promise( function(resolve, reject) {  
    setTimeout( () => reject(), 1000);  
})  
.then( () => {  
    console.log("Se ha resuelto la promesa");  
})  
.catch( () => {  
    console.log("Ha habido un error");  
})
```

Si tenemos varias promesas encadenadas, la función `catch` se puede poner al final y se capturará el error cuando cualquiera de las promesas falle.

```
fetch('https://swapi.dev/api/people/1')  
  .then( response => response.json() )  
  .then( data => console.log(data) )  
  .catch( () => console.log("Ha habido un error") );
```

# 3

async/await



Hay una sintaxis alternativa para trabajar con promesas llamada **`async/await`** que se basa en el uso de esas dos palabras clave.

La palabra clave **`async`** siempre se pone delante de una función e indica que dicha función siempre devolverá una promesa. Cualquier valor que devuelva la función será envuelto automáticamente en una promesa.

```
async function f(){  
  return 1;  
}
```

```
console.log(f());
```



Filtrar salida

```
► Promise { <state>: "fulfilled", <value>: 1 }
```

Dentro de una función `async` podemos utilizar la palabra clave **`await`** cuya sintaxis es la siguiente:

```
let value = await promise;
```

**`await`** hace que la ejecución de la función se detenga hasta que la promesa se haya resuelto.

```
async function f() {  
  let promise = new Promise( (resolve, reject) => {  
    setTimeout( () => resolve("Hecho"), 1000 );  
  });  
  
  let result = await promise;  
  
  console.log(result);  
}  
f();
```

Cuando usamos la palabra clave **`await`**, hay dos posibilidades para gestionar que la promesa sea rechazada:

## Bloque `try..catch`

La primera posibilidad es capturarlo en un bloque `try..catch` tal como se gestionaría cualquier otra excepción.

```
async function f() {  
  try {  
    let response = await fetch("http://no-existe");  
  } catch (err) {  
    console.log(err);  
  }  
};  
  
f();
```

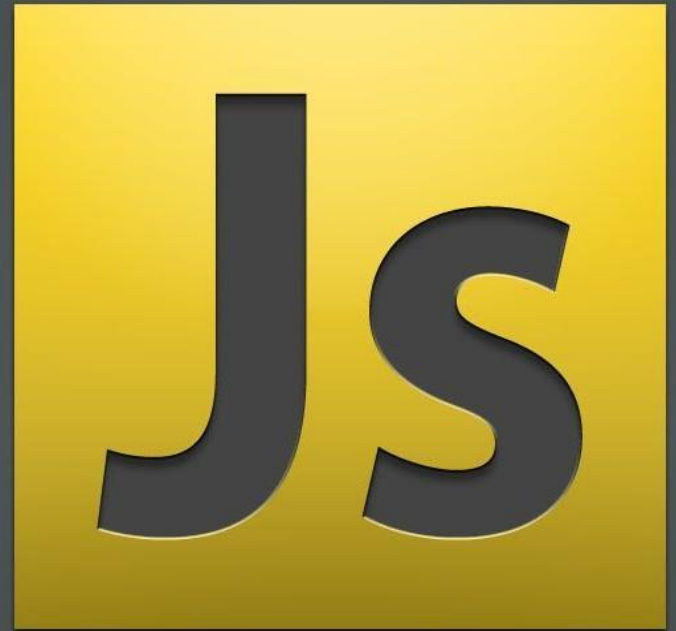
## Con `.catch()`

Si no hay un `try..catch`, la promesa generada en la función `async` será rechazada, por lo que podríamos capturarla con un `catch`

```
async function f() {  
    let response = await fetch("http://no-existe");  
};  
  
f().catch( err => console.log(err) );
```

# 4

## PROMISE API





La clase **Promise** dispone de 6 **métodos estáticos** que pueden ser muy útiles cuando trabajamos con promesas.

- Promise.all
- Promise.allSettled
- Promise.race
- Promise.any
- Promise.resolve
- Promise.reject

## Promise.all

Toma un iterable (*usualmente un array de promesas*) y devuelve una nueva promesa que es devuelta cuando todas las promesas listadas se resuelvan.

```
Promise.all([
  new Promise(resolve => setTimeout(() => resolve(1), 3000)),
  new Promise(resolve => setTimeout(() => resolve(2), 2000)),
  new Promise(resolve => setTimeout(() => resolve(3), 1000))
]).then(alert);
```

## Otro ejemplo

```
let urls = [  
  'https://api.github.com/users/iliakan',  
  'https://api.github.com/users/remy',  
  'https://api.github.com/users/jeresig'  
];  
  
// "mapeamos" cada url a la promesa de su fetch  
let requests = urls.map(url => fetch(url));  
  
// Promise.all espera hasta que todas la tareas estén resueltas  
Promise.all(requests)  
  .then(responses => responses.forEach(  
    response => alert(`${response.url}: ${response.status}`)  
  ));
```

## Promise.allSettled

Promise.all solo se resuelve si todas sus promesas finalizan con éxito.

**Promise.allSettled** solo espera a que todas las promesas se resuelvan sin importar sus resultados. En este caso devuelve un array que tiene:

```
{status:"fulfilled", value:result} para respuestas exitosas,  
{status:"rejected", reason:error} para errores.
```

```
let urls = [  
  'https://api.github.com/users/iliakan',  
  'https://api.github.com/users/remy',  
  'https://no-such-url'  
];  
  
Promise.allSettled(urls.map(url => fetch(url)))  
  .then(results => {  
    results.forEach((result, num) => {  
      if (result.status == "fulfilled") {  
        alert(`${urls[num]}: ${result.value.status}`);  
      }  
      if (result.status == "rejected") {  
        alert(`${urls[num]}: ${result.reason}`);  
      }  
    });  
  });
```

## Promise.race

Similar a Promise.all, pero espera solamente por la primera respuesta y obtiene su resultado (o error)

```
Promise.race([
  new Promise((resolve, reject) => setTimeout(() =>
    resolve(1), 1000)),
  new Promise((resolve, reject) => setTimeout(() =>
    reject(new Error("Whoops!")), 2000)),
  new Promise((resolve, reject) => setTimeout(() =>
    resolve(3), 3000))
]).then(alert); // 1
```

Aquí se resolvería solo la primera promesa, por lo que el resto se ignorarían.

## Promise.any

Similar al anterior, pero espera por la primera **promesa cumplida**.

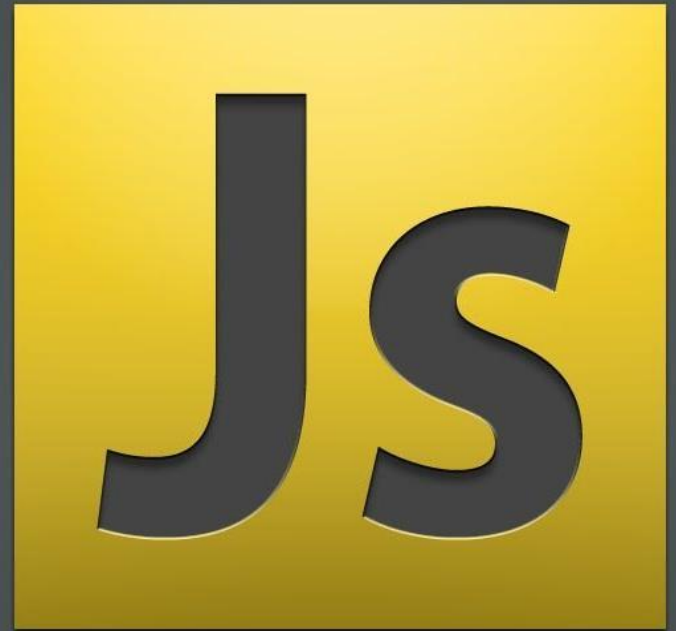
## Promise.resolve / Promise.reject

Permiten crear una promesa resuelta o rechazada respectivamente. Apenas son utilizados.



5

FETCH



Actualmente, el método utilizado en JavaScript para realizar peticiones de red es **fetch**, que ha reemplazado al antiguo **XMLHttpRequest**.

Su sintaxis básica es:

```
let promise = fetch(url, [options])
```

Donde *url* es la dirección URL a la que se desea acceder y *options* los parámetros opcionales, como puede ser el método a utilizar o los encabezados de la petición.

Este método **devuelve una promesa**.

Ejemplo de solicitud.

```
const url = 'https://swapi.dev/api/people/1';

async function getData() {
  console.log(url);
  let response = await fetch( url );
  console.log(response);
}

getData();
```

Algo importante es que la promesa devuelta por fetch resuelve la respuesta con un objeto de tipo **Response**, que **contiene los encabezados de la petición**.

```
▼ Response { type: "cors", url: "https://swapi.dev/api/people/1", redirected: false, status: 200, ok:
Headers(1), body: ReadableStream, bodyUsed: false }
  ► body: ReadableStream { locked: false }
    bodyUsed: false
  ▼ headers: Headers { "content-type" → "application/json" }
    ► <entries>
    ► <prototype>: HeadersPrototype { append: append(), delete: delete(), get: get(), ... }
    ok: true
    redirected: false
    status: 200
    statusText: "OK"
    type: "cors"
    url: "https://swapi.dev/api/people/1"
  ► <prototype>: ResponsePrototype { clone: clone(), arrayBuffer: arrayBuffer(), blob: blob(), ... }
```

Las propiedades más interesantes de este objeto son **status** y **ok**, que contienen el estado de la respuesta y un booleano que será true si el estado es 200 a 299.

- **1xx:** Mensaje informativo.
- **2xx:** Exito
  - 200 OK
  - 201 Created
  - 202 Accepted
  - 204 No Content
- **3xx:** Redirección
  - 300 Multiple Choice
  - 301 Moved Permanently
  - 302 Found
  - 304 Not Modified
- **4xx:** Error del cliente
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- **5xx:** Error del servidor
  - 500 Internal Server Error
  - 501 Not Implemented
  - 502 Bad Gateway
  - 503 Service Unavailable

Si la respuesta es exitosa, debemos utilizar un segundo método para obtener el cuerpo de la respuesta.

Este método varía en función del formato de la misma.

- `response.text()`: en formato texto
- `response.json()`: la devuelve como un JSON
- `response.formData()`: la devuelve como objeto FormData
- `response.blob()`: objeto de tipo Blob (datos binarios)
- `response.arrayBuffer()`: representación de datos de bajo nivel (arrayBuffer)

Obteniendo el cuerpo de la solicitud con **await**.

```
const url = 'https://swapi.dev/api/people/1';

async function getData() {
  let response = await fetch( url, );
  let json = await response.json();
  console.log(json);
}
getData();
```

## Alternativa usando promesas

```
const url = 'https://swapi.dev/api/people/1';

async function getData() {
  fetch( url )
    .then( response => response.json() )
    .then( data => console.log(data) )
}
getData();
```

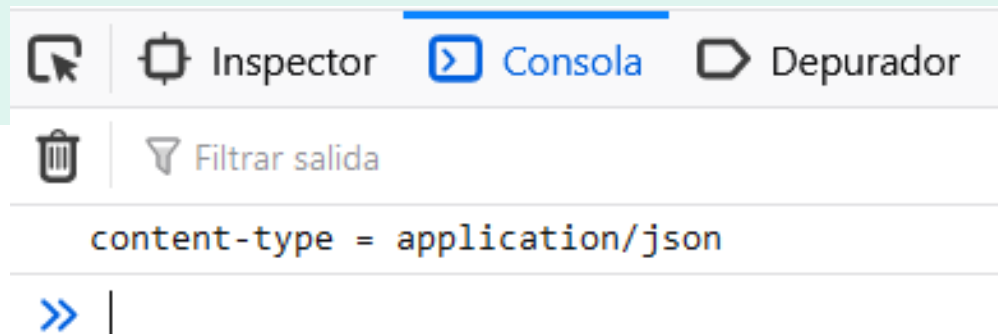


Se pueden ver los encabezados HTTP de respuesta dentro de **response.headers**

```
const url = 'https://swapi.dev/api/people/1';

async function getData() {
  fetch( url )
    .then( response => {
      for (let [key, value] of response.headers) {
        console.log(`${key} = ${value}`);
      }
    } )
}

getData();
```



Alternativamente tambín podemos indicar los encabezados que queremos ańadir a la petici3n.

Esto se indica en el segundo parámetro de **fetch**

```
let response = fetch(protectedUrl, {  
  headers: {  
    Authentication: 'secret'  
  }  
});
```

Si queremos realizar una **petición POST**, lo podremos indicar también en las opciones.

```
let user = {  
  nombre: 'Juan',  
  apellido: 'Perez'  
};  
  
let response = await fetch('/article/fetch/post/user', {  
  method: 'POST',  
  headers: {  
    'Content-Type': 'application/json;charset=utf-8'  
  },  
  body: JSON.stringify(user)  
});  
  
let result = await response.json();  
console.log(result.message);
```

Observa como utilizamos **JSON.stringify()** para convertir el objeto con los datos en una cadena de texto.

## JSON.stringify()

El método `JSON.stringify()` convierte un objeto o valor de JavaScript en una cadena de texto JSON, opcionalmente reemplaza valores si se indica una función de reemplazo, o si se especifican las propiedades mediante un array de reemplazo.

## Pruébalo

### JavaScript Demo: JSON.stringify()

```
1 console.log(JSON.stringify({ x: 5, y: 6 }));  
2 // expected output: '{"x":5,"y":6}'  
3  
4 console.log(JSON.stringify([new Number(3), new String('false'), new Boolean(false)]));  
5 // expected output: '[3,"false",false]'  
6  
7 console.log(JSON.stringify({ x: [10, undefined, function(){}], Symbol('')} ));  
8 // expected output: '{"x":[10,null,null,null]}'  
9
```

Si quisiéramos realizar el paso opuesto, convertir una cadena JSON en objetos, podemos usar la función **JSON.parse()**

## JSON.parse()

### Resumen

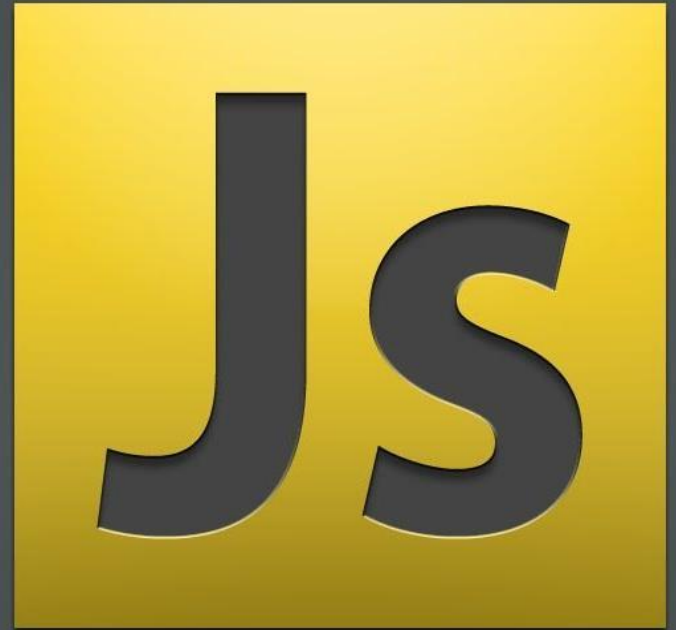
El método `JSON.parse()` analiza una cadena de texto como JSON, transformando opcionalmente el valor producido por el análisis.

### Sintaxis

```
JSON.parse(text[, reviver])
```

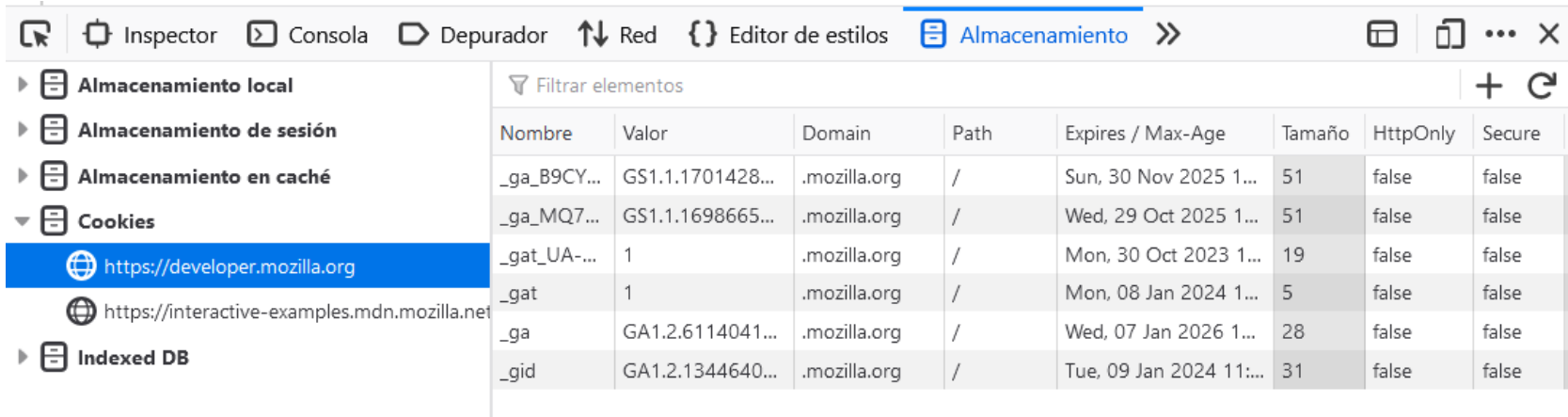
# 6

ALMACENAMIENTO:  
COOKIES



Las **cookies** son pequeñas cadenas de datos que se almacenan en el navegador.

Podemos verlas en la pestaña *Almacenamiento* de *Herramientas del desarrollador*



The screenshot shows the Chrome DevTools Storage tab. The left sidebar lists storage types: Almacenamiento local, Almacenamiento de sesión, Almacenamiento en caché, Cookies (selected), and Indexed DB. The main pane shows a table of cookies for the selected site. The table has columns: Nombre, Valor, Domain, Path, Expires / Max-Age, Tamaño, HttpOnly, and Secure.

Nombre	Valor	Domain	Path	Expires / Max-Age	Tamaño	HttpOnly	Secure
_ga_B9CY...	GS1.1.1701428...	.mozilla.org	/	Sun, 30 Nov 2025 1...	51	false	false
_ga_MQ7...	GS1.1.1698665...	.mozilla.org	/	Wed, 29 Oct 2025 1...	51	false	false
_gat_UA-...	1	.mozilla.org	/	Mon, 30 Oct 2023 1...	19	false	false
_gat	1	.mozilla.org	/	Mon, 08 Jan 2024 1...	5	false	false
_ga	GA1.2.6114041...	.mozilla.org	/	Wed, 07 Jan 2026 1...	28	false	false
_gid	GA1.2.1344640...	.mozilla.org	/	Tue, 09 Jan 2024 11:...	31	false	false

Son **propias del dominio**, es decir, cada dominio puede establecer sus propias cookies y únicamente las páginas de ese mismo dominio podrán acceder a ellas.

Las cookies se establecen cuando el servidor utiliza la cabecera de respuesta **Set-Cookie**.

En cada solicitud del navegador al servidor, agregará automáticamente las cookies en la cabecera **Cookie**.

2 solicitudes | 179 B / 1,24 KB transferido | Finalizado: 383 ms



Son **propias del dominio**, es decir, cada dominio puede establecer sus propias cookies y únicamente las páginas de ese mismo dominio podrán acceder a ellas.

Las cookies se establecen cuando el servidor utiliza la cabecera de respuesta **Set-Cookie**.

En cada solicitud del navegador al servidor, agregará automáticamente las cookies en la cabecera **Cookie**.

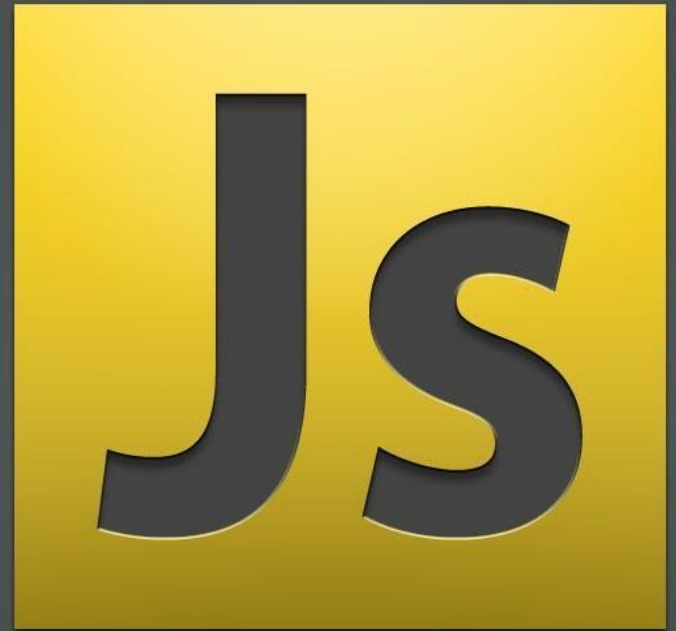
The screenshot shows the Chrome DevTools Network tab. A list of requests is visible on the left, with the first one (GET) selected. The 'Headers' pane on the right shows the 'Request Headers' for the selected request. The 'Cookie' header is highlighted, showing a long string of cookies. The status bar at the bottom indicates 2 requests, 179 B / 1,24 KB transferred, and a final time of 383 ms.

Esta...	Mét...	Dominio	Archivo	Iniciador	Tipo	Transferido	Tam...
200	GET	develope...	whoami	main.70c8cb...	json	479 B	179 B
200	POST	develope...	d3efc897-a1af-401a-ab35-73f0945d66c	main.70c8cb...	plain	763 B	0 B

2 solicitudes | 179 B / 1,24 KB transferido | Finalizado: 383 ms

# 7

ALMACENAMIENTO:  
LOCAL STORAGE Y  
SESSION STORAGE



Los objetos **localStorage** y **sessionStorage** permiten guardar pares clave/valor.

Lo más relevante es que los datos sobreviven a una recarga de la página (**sessionStorage**) y hasta al reinicio del ordenador (**localStorage**).

Diferencias de ambos respecto a las cookies son:

- No se envían automáticamente al servidor
- El servidor no puede manipularlos vía cabeceras HTTP, todo se hace con JavaScript
- El almacenaje está vinculado al origen (tripleto dominio/protocolo/puerto)

Los métodos disponibles son:

`setItem(clave, valor)` – almacenar un par clave/valor.

`getItem(clave)` – obtener el valor por medio de la clave.

`removeItem(clave)` – eliminar la clave y su valor.

`clear()` – borrar todo.

`key(índice)` – obtener la clave de una posición dada.

`length` – el número de ítems almacenados.

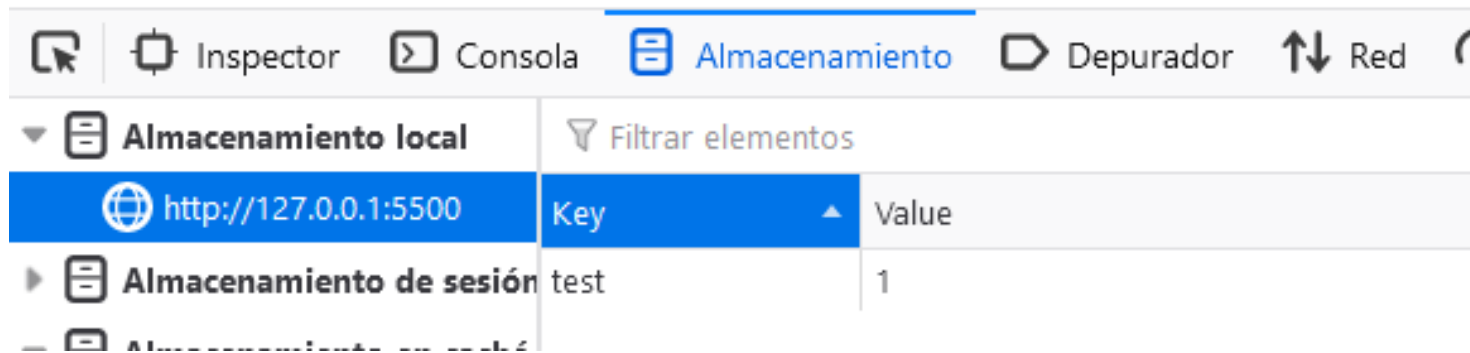
## Local Storage

Las principales funcionalidades del **localStorage** son:

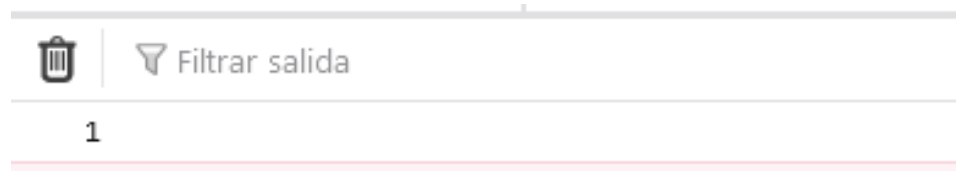
- Es compartido entre todas las pestañas y ventanas del mismo origen.
- Los datos no expiran. Persisten a los reinicios del navegador y del sistema operativo

**setItem** y **getItem** permiten escribir y recuperar valores.

```
localStorage.setItem('test', 1);
```

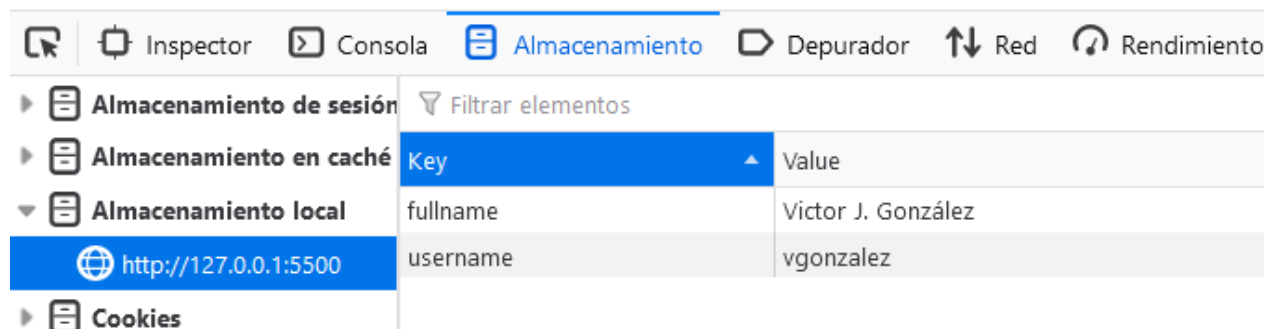


```
console.log(localStorage.getItem('test'));
```



**removeItem()** elimina el valor que se indique y **clear()** borra todos los valores.

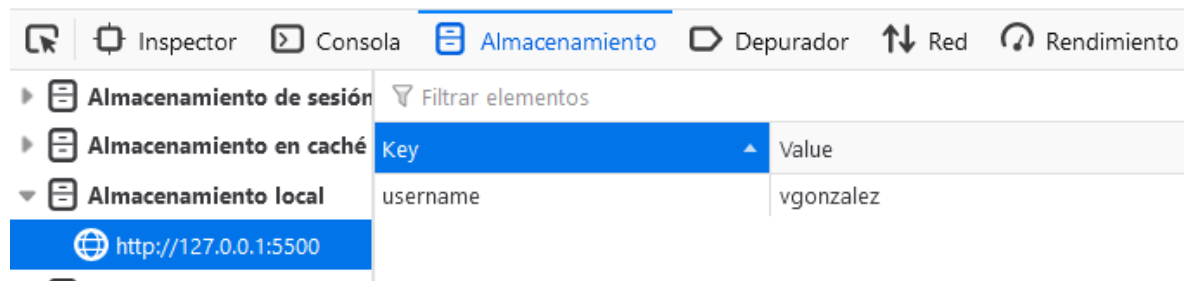
```
localStorage.setItem('fullname', 'Victor');  
localStorage.setItem('username', 'vgonzalez');
```



The screenshot shows the Chrome DevTools Storage tab. The left sidebar lists 'Almacenamiento de sesión', 'Almacenamiento en caché', 'Almacenamiento local', and 'Cookies'. The 'Almacenamiento local' section is expanded, showing a table of items. The table has two columns: 'Key' and 'Value'. The first row shows 'fullname' with the value 'Victor J. González'. The second row shows 'username' with the value 'vgonzalez'.

Key	Value
fullname	Victor J. González
username	vgonzalez

```
console.log(localStorage.removeItem('fullname'));
```



The screenshot shows the Chrome DevTools Storage tab after the 'fullname' item has been removed. The 'Almacenamiento local' section is still expanded, but the table now only contains one row: 'username' with the value 'vgonzalez'.

Key	Value
username	vgonzalez



Los objetos de almacenaje **no son iterables**, por lo que la única forma de iterar sobre todos sus elementos es utilizar iteración usando el índice mediante la propiedad **length** y la función **key()**.

```
localStorage.setItem('fullname', 'Victor J. González');
localStorage.setItem('username', 'vgonzalez');

for(let i=0; i<localStorage.length; i++) {
  let key = localStorage.key(i);
  console.log(`${key}: ${localStorage.getItem(key)}`);
}
```



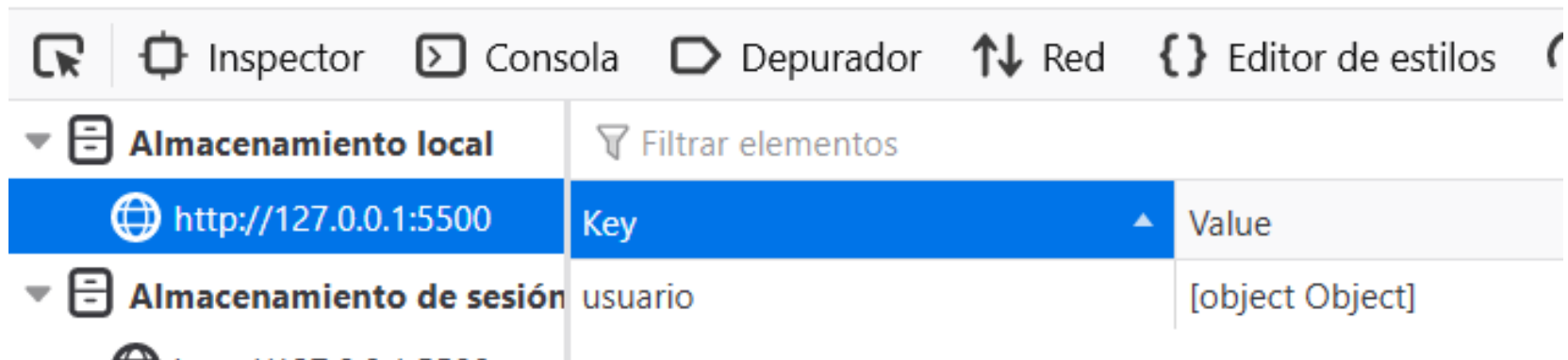
Filtrar salida

username: vgonzalez

fullname: Victor J. González

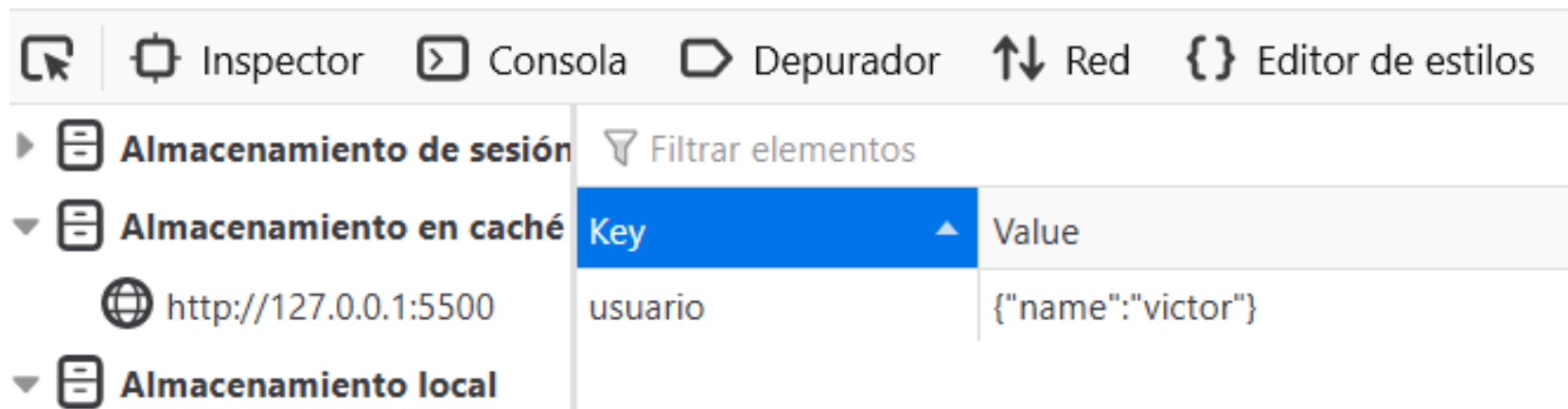
Algo importante es que en el localStorage **solo se pueden guardar *strings***, por lo que cualquier otro tipo de datos se convertirá automáticamente a una cadena.

```
localStorage.usuario = { name: "victor"};
```



Si queremos guardar JSON deberemos utilizar la función **JSON.stringify()** para convertirlo adecuadamente a una cadena.

```
localStorage.usuario = JSON.stringify(  
    { name: "victor"});
```



## Session Storage

Características:

- Mismos métodos y características que localStorage
- Solo existe dentro de la pestaña del navegador
  - Otra pestaña con la misma página tendrá un contenido distinto
  - Sí se comparte entre iframes del mismo origen de la misma pestaña
- Los datos sobreviven a un refresco de página, pero no a cerrar/abrir la pestaña

## Evento storage

Hay un evento asociado al *localStorage* y *sessionStorage* denominado **storage** que se dispara cada vez que hay cambios en el *localStorage*.

Sus propiedades son:

`key` – la clave que ha cambiado, (`null` si se llama `.clear()`).

`oldValue` – el anterior valor (`null` si se añade una clave).

`newValue` – el nuevo valor (`null` si se borra una clave).

`url` – la url del documento donde ha pasado la actualización.

`storageArea` – bien el objeto `localStorage` o `sessionStorage`,  
donde se ha producido la actualización.

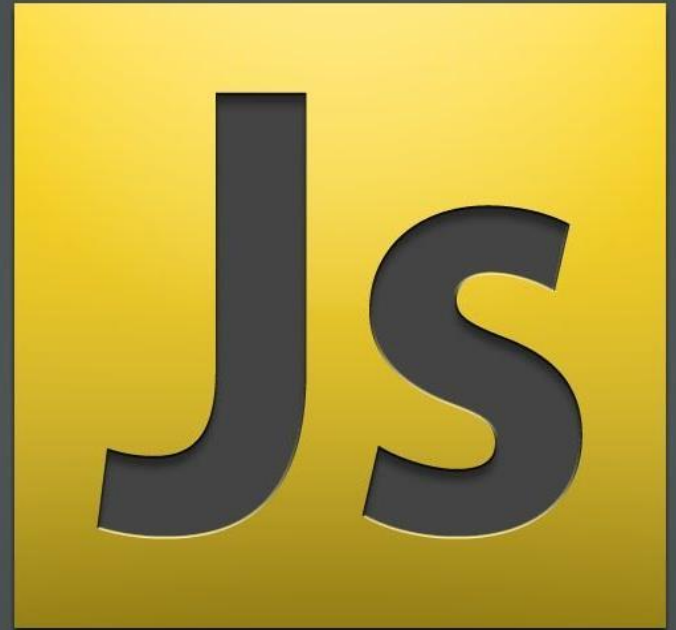
Este evento se dispara en todos los objetos window donde el almacenaje es accesible, **excepto en el que lo ha causado.**

```
window.onstorage = e => {  
  if (e.key !== 'now') return;  
  console.log(`${e.key}: ${e.newValue} en ${e.url}`)  
};  
  
localStorage.setItem('now', Date.now());
```

Este mecanismo es útil para que ventanas del mismo origen **puedan intercambiar mensajes.**

# 7

AUTENTICACIÓN:  
JSON WEB  
TOKEN (JWT)



Tradicionalmente, el mantenimiento de la sesión en aplicaciones Web se ha realizado con **cookies**.

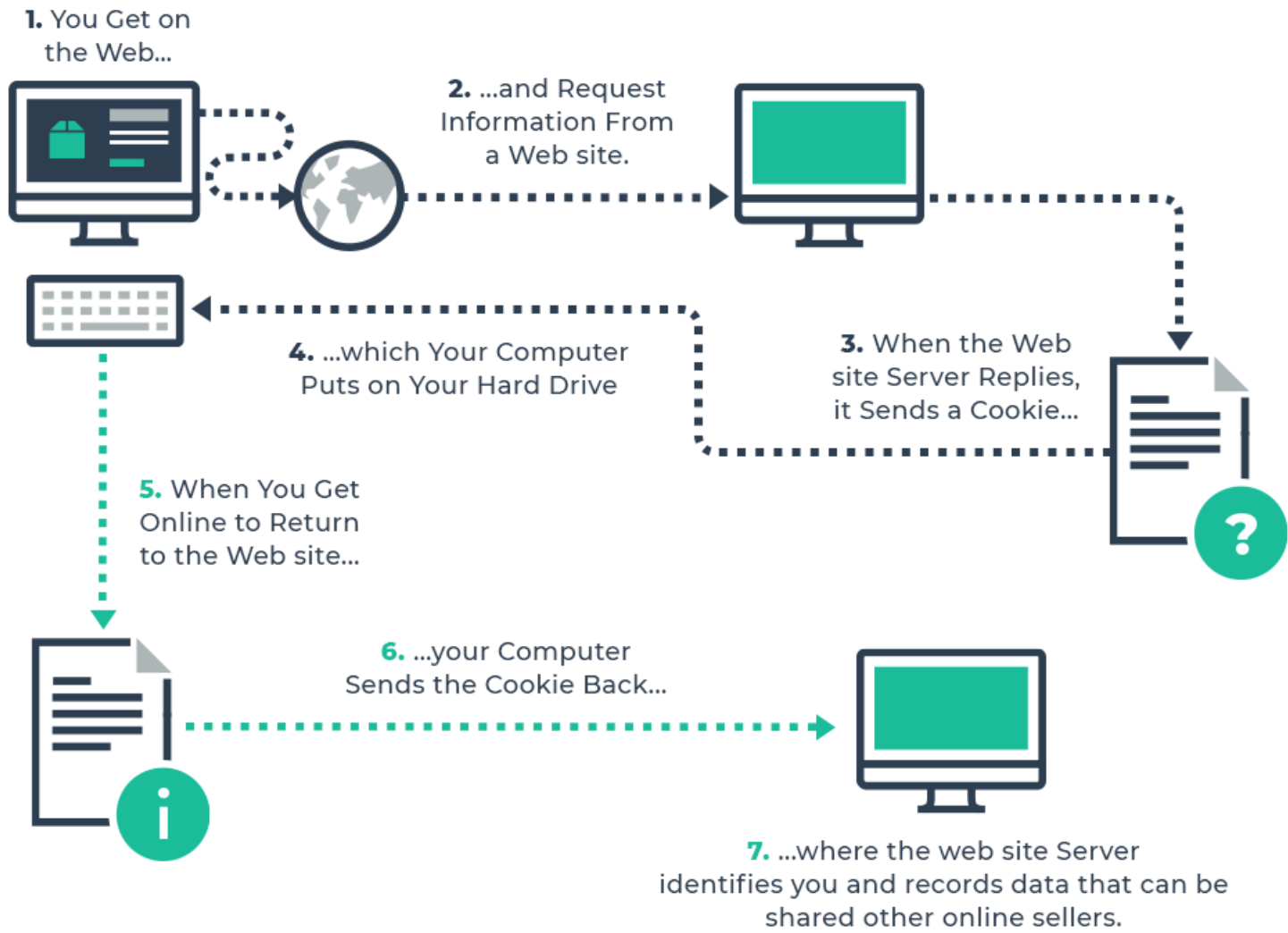
Una **cookie** es un par nombre-valor que está asociada a un dominio.

El mecanismo de las cookies es muy sencillo:

- El servidor envía la cookie al navegador en la respuesta, donde es almacenada.
- En cualquier petición que se realice al servidor desde el cliente se envía **automáticamente** la cookie







Con las cookies podemos comprobar si el usuario ha iniciado sesi3n, pero son exclusivas del servidor que las ha generado, por lo que la sesi3n no se puede mantener cuando hay consultas a varios servidores.

Esto puede ser un problema cuando trabajamos con APIs, especialmente sobre **microservicios**, que pueden estar repartidos en mltiples servidores.

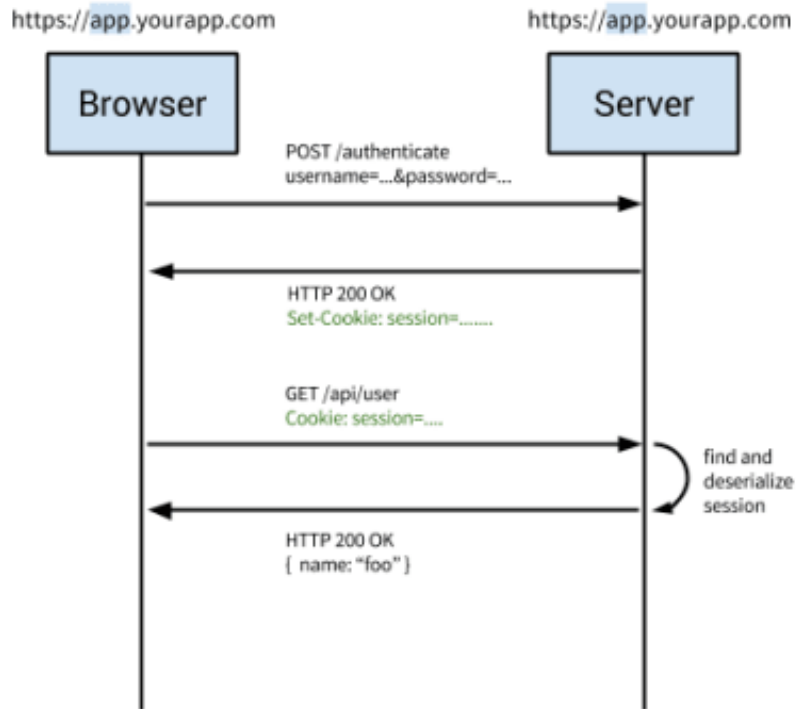
La alternativa es el uso de JSON Web Token (JWT).

Un JWT es simplemente un **token** cuya posesión acredita que el usuario es quien dice ser:

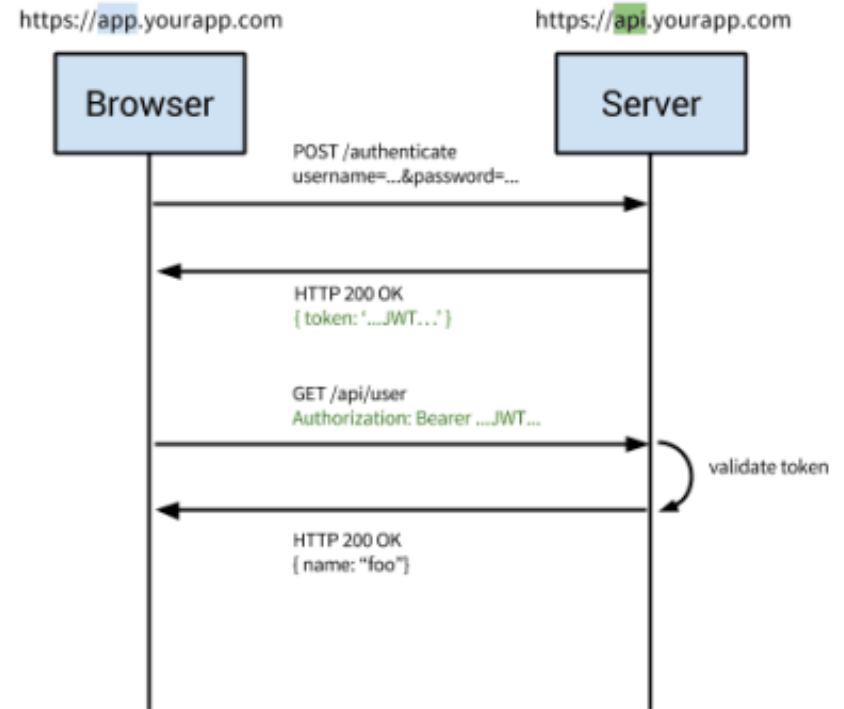
- El usuario se autentica en el servidor, el cual le devuelve el token.
- Cada vez que el usuario realice una petición debe adjuntar manualmente el token para acreditar su identidad.

Es importante destacar que ni las cookies ni JWT son mecanismos de autenticación, sino que son utilizados después de la autenticación para acreditar al usuario.

## Traditional Cookie-Based Auth



## Modern Token-Based Auth



La autenticación basada en JWT **carece de estado (*stateless*)**, ya que el servidor no guarda información de qué usuarios hay conectados ni almacena los tokens que se han emitido.

Normalmente los tokens se envían como un *Authorization header* con el valor *Bearer {JWT}*, pero también se pueden enviar en el cuerpo de una petición POST o incluso como *query parameter*.

Los pasos son:

- El usuario envía sus credenciales
- El servidor verifica su validez y devuelve un token firmado
- El token se guarda en el cliente en el *local storage*, el *session storage*, en memoria o incluso como una *cookie*.
- Cualquier petición a partir de ahora incluirá ese token
- El servidor decodifica el JWT y verifica si el token es válido
- Cuando el usuario se desconecta el token es eliminado en el lado del cliente.

Todo lo relativo a JWT está definido en el estándar RFC 7519.

Un token JWT consta de tres partes, codificadas en Base64 y separadas por puntos.

**HEADER.PAYLOAD.SIGNATURE**



## Header

Consta generalmente de dos valores: el tipo de token y el algoritmo de firma o cifrado que se ha utilizado.

```
{ "alg": "HS256", "typ": "JWT" }
```

Algoritmos:

- **HS256**: HMAC-SHA256
- **RS256**: RSA con SHA-256
- **ES256**: ECDSA con SHA-256

## Payload

Contiene la información real que se transmitirá a la aplicación. Se indica como pares clave-valor denominadas **claims**, que pueden ser de tres tipos:

- **Claims registrados:** que figuran en el [IANA JSON Web Token Claim Register](#).
- **Claims públicos:** pueden definirse a voluntad, pero hay que registrarlos en el IANA para evitar conflictos.
- **Claims privados:** cuando los utilizamos en comunicaciones entre nuestras propias aplicaciones.

Todos los *claims* son opcionales, debiendo limitarse a la información estrictamente necesaria.

Ejemplo:

```
{  
  "name": "victor",  
  "admin": "true",  
  "exp": "1674453570"  
}
```


## Signature

Se crea utilizando una codificación **Base64** del *header* y del *payload*, así como el método de firma o cifrado.

Es necesaria una **clave secreta** únicamente conocida por la aplicación original.

La firma garantiza que el mensaje no haya sido modificado.

Hay múltiples librerías en JavaScript para crear el JWT. Por ejemplo, **jsonwebtoken**.


jsonwebtoken 

9.0.0 • Public • Published a month ago

 Readme

 Code Beta

 4 Dependencies

 22.232 Dependents

 79 Versions

## jsonwebtoken

Build	Dependency
 build passing	Dependency Status

An implementation of **JSON Web Tokens**.

This was developed against `draft-ietf-oauth-json-web-token-08`. It makes use of **node-jws**


## Install

```
$ npm install jsonwebtoken
```

### Install

```
> npm i jsonwebtoken
```

### Repository

 [github.com/auth0/node-jsonwebtoken](https://github.com/auth0/node-jsonwebtoken)

### Homepage

 [github.com/auth0/node-jsonwebtoken#...](https://github.com/auth0/node-jsonwebtoken#readme)

### Weekly Downloads

11.653.880



### Version

9.0.0

### License

MIT

En el servidor:

```
const jwt = require('jsonwebtoken');
const secret = 'S3cr3T';

app.post("/login", (req, res) => {
  const {username, pass} = req.body;
  // Aquí comprobaría si las credenciales son válidas
  const token = generateAccessToken({
    username: req.body.username });
  res.json(token);
})

app.get("/test", (req, res) => {
  const headers = req.headers;
  let decoded = jwt.verify(headers.authorization.split('
')[1], secret);
  console.log(decoded);
})
```

En el cliente: solicitud de login para obtener jwt

```
fetch('http://localhost:5000/login', {  
  method: 'POST',  
  headers: {  
    'Content-Type': 'application/json'  
  },  
  body: JSON.stringify({  
    username: 'victor',  
    pass: '1234'  
  })  
})  
  .then( response => response.json())  
  .then( jwt => {  
    // Se almacenaría de alguna manera  
    console.log(jwt);  
  });
```

En el cliente: llamada adjuntando el token

```
fetch('http://localhost:5000/test', {  
  headers: {  
    Authorization: `Bearer ${jwt}`  
  }  
})  
  .then( response => console.log(response));
```



https://github.com/firebase/php-jwt

Introducción al curso |... Index - Prácticas de D... DWECC ISO DWECC - Proyecto

phpunitxml:dist feat: add cached keyset (#557) 5 months ago

☰ README.md

Test Suite **passing** stable **v6.3.2** downloads **181.21 M** license **BSD-3-Clause**

## PHP-JWT

A simple library to encode and decode JSON Web Tokens (JWT) in PHP, conforming to [RFC 7519](#).

### Installation

Use composer to manage your dependencies and download PHP-JWT:

```
composer require firebase/php-jwt
```

Optionally, install the `paragonie/sodium_compat` package from composer if your php is < 7.2 or does not have libsodium installed:

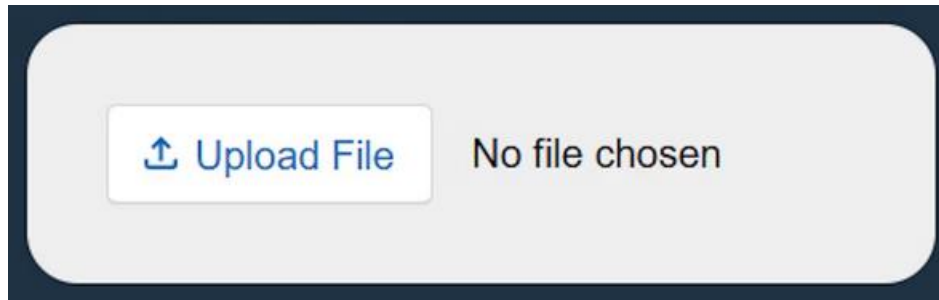
# 8

CARGA DE  
FICHEROS



Desde una aplicación web **no podremos acceder al sistema de ficheros del equipo cliente**, pero hay ocasiones en que necesitaremos acceder a ficheros seleccionados por el usuario.

La forma para hacerlo es interceptando el elemento HTML **`<input type="file">`**.



Para ello solo hemos de capturar el evento **change** del elemento input.

```
<label for="input-gpx">Fichero GPX</label>  
<input type="file" id="input-gpx" name="input-gpx">
```

```
const inputGPX = document.querySelector('#input-gpx');  
  
inputGPX.addEventListener( 'change' , (e) => {  
    e.preventDefault();  
});
```

Una vez que el usuario haya seleccionado el archivo ya lo tendremos disponible en Javascript.

Podremos acceder al fichero o ficheros que haya seleccionado el usuario mediante la propiedad **files**.

```
inputGPX.addEventListener( 'change' , (e) => {  
    const file = inputGPX.files[0];
```

El contenido de files es un objeto de tipo **FileList**, ya que puede tener múltiples elementos, por ello accedemos al primer elemento del mismo.

Para acceder al contenido del fichero recurriremos al objeto **FileReader**.

Primero creamos una instancia de `FileReader` y luego, dado que es un fichero de texto, utilizamos la función **`readAsText()`**.

```
if (file) {  
    const reader = new FileReader();  
    const r = reader.readAsText(file);  
}
```

En este ejemplo compruebo primero que el fichero tenga contenido, pero podríamos hacer más comprobaciones, por ejemplo, que el fichero tenga una extensión determinada.

El problema aquí es que la lectura del fichero no es una tarea síncrona, por lo que debemos esperar al evento de finalización de lectura del fichero.

```
reader.onload = (event) => {
```

En la página de documentación de FileReader puedes ver información sobre otros eventos que pueden ser útiles para hacer un seguimiento de la carga del fichero, por ejemplo, cuando comienza (**onloadstart**), si hay un error (**onerror**) o para hacer un seguimiento del progreso de carga (**onprogress**)

Ya solo queda leer el contenido accediendo a la propiedad **result** del elemento input.

```
reader.onload = (event) => {  
    const fileContent = event.target.result;
```



Vamos a ver ahora cómo haríamos para leer un **fichero GPX** seleccionado por el usuario y convertirlo a **JSON** para realizar algún tipo de manipulación con él.

Lo primero que necesitaremos es incluir una librería que realice por nosotros el proceso de conversión.

Vamos a utilizar una que se llama **toGeoJSON** (<https://github.com/mapbox/togeojson>)

### Convert KML and GPX to GeoJSON.

This converts [KML](#) & [GPX](#) to [GeoJSON](#), in a browser or with [Node.js](#).

- ☒ Tiny
- ☒ Tested
- ☒ Node.js + Browsers

Want to use this with [Leaflet](#)? Try [leaflet-omnivore](#)!

Como muchas librerías se puede utilizar en el lado del servidor (con Node) o en el lado del cliente.

## Node.js

Install it into your project with `npm install --save @mapbox/togeojson`.

```
// using togeojson in nodejs

var tj = require('@mapbox/togeojson'),
    fs = require('fs'),
    // node doesn't have xml parsing or a dom. use xmldom
    DOMParser = require('xmldom').DOMParser;
```



## Browser

Download it into your project like

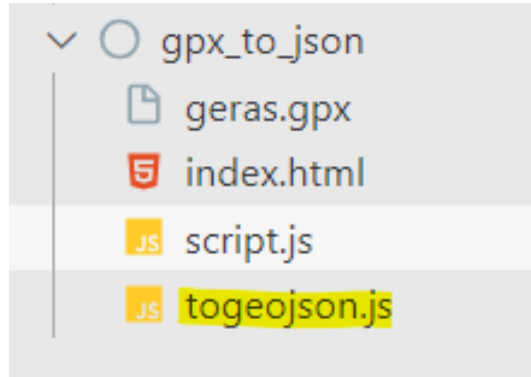
```
wget https://raw.githubusercontent.com/mapbox/togeojson/master/togeojson.js
```



```
<script src='jquery.js'></script>
```



Seguimos las instrucciones para el lado del cliente. Descargamos el fichero de la librería y lo incluimos en nuestro HTML.



```
<script src='togejson.js'></script>  
<script src="script.js"></script>
```

La función de la librería que necesitamos es **toGeoJSON.gpx()**, por lo que vamos a ver la ayuda

### **toGeoJSON.gpx(doc)**

Convert a GPX document to GeoJSON. The first argument, `doc`, must be a GPX document as an XML DOM - not as a string. You can get this using jQuery's default `.ajax` function or using a bare XMLHttpRequest with the `.response` property holding an XML DOM.

The output is a JavaScript object of GeoJSON data, same as `.kml` outputs.

La ayuda hace referencia a jQuery, por lo que nos es poco útil, pero algo importante que nos dice es que el parámetro de esta función no puede ser una cadena, sino que tiene que ser un XML DOM.

Esto quiere decir que tenemos que parsear el documento GPX, lo cual haremos con el objeto **DOMParser**

```
const parser = new DOMParser();  
const xmlDoc = parser.parseFromString(fileContent,  
                                     'application/xml');
```

Esta función devolverá un objeto de la clase XMLDocument, que sí que podremos pasar a la función gpx()

```
const json = toGeoJSON.gpx(xmlDoc);
```

Y ya tenemos el JSON con el que podemos trabajar.

Ahora que tenemos el JSON, pero nos falta saber cómo podemos trabajar con él.

Los ficheros GeoJSON tienen una estructura definida en el RFC 7946 (<https://geojson.org/>)

## GEOJSON

GeoJSON is a format for encoding a variety of geographic data structures.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

GeoJSON supports the following geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon. Geometric objects with additional properties are Feature objects. Sets of features are contained by FeatureCollection objects.

### The GeoJSON Specification (RFC 7946)

También podemos ir analizando los datos devueltos por la función `gpx()` para inferir cómo es su estructura.

```
const json = toGeoJSON.gpx(xmlDoc);  
console.log(json);
```


Es un objeto con 2 propiedades:  
*features* y *type*

```
▼ Object { type: "FeatureCollection", features: (1) [...] }  
  ▼ features: Array [ {...} ]  
    ▼ 0: Object { type: "Feature", properties: {...}, geometry: {  
      ► geometry: Object { type: "LineString", coordinates: (36  
      ► properties: Object { name: "Carrera de montaña a la hor  
        type: "Feature"  
      ► <prototype>: Object { ... }  
      length: 1  
      ► <prototype>: Array []  
      type: "FeatureCollection"  
      ► <prototype>: Object { ... }
```

features contiene un array, pero con un solo elemento, vamos a quedarnos solo con él

```
// console.log(xmlDoc);  
const json = toGeoJSON.gpx(xmlDoc);  
console.log(json.features[0]);
```

Ahora tenemos un objeto con tres propiedades: geometry, properties y type



```
▼ Object { type: "Feature", properties: {...}, geometry: {...} }  
  ► geometry: Object { type: "LineString", coordinates: (3632) [...] }  
  ► properties: Object { name: "Carrera de montaña a la hora del almuerzo",  
    type: "Feature"  
  }  
  ► <prototype>: Object { ... }
```



```
// console.log(xmlDoc);  
const json = toGeoJSON.gpx(xmlDoc);  
console.log(json.features[0]);
```

Nos quedamos con **geometry**. Observa que tiene una propiedad llamada **coordinates** que contiene un array con muchos elementos

```
▼ Object { type: "Feature", properties: {...}, geometry: {...} }  
  ▼ geometry: Object { type: "LineString", coordinates: (3632) [...] }  
    ▼ coordinates: Array(3632) [ (3) [...], (3) [...], (3) [...], ... ]  
      ▼ [0...99]  
        ► 0: Array(3) [ -5.549981, 42.596538, 819 ]  
        ► 1: Array(3) [ -5.549972, 42.596534, 819 ]  
        ► 2: Array(3) [ -5.549964, 42.59653, 819 ]  
        ► 3: Array(3) [ -5.549955, 42.596526, 819 ]  
        ► 4: Array(3) [ -5.549937, 42.596518, 819 ]  
        ► 5: Array(3) [ -5.549907, 42.596511, 819 ]
```

Cada elemento del array contiene tres elementos que tienen pinta de ser la longitud, latitud y altitud

Vamos ahora  
con **properties**

Fíjate que el número de elementos de estos arrays corresponde con el número de puntos del track, por lo que habrá una correspondencia entre uno y otro.

```
▼ Object { type: "Feature", properties: {...}, geometry: {...} }
  ► geometry: Object { type: "LineString", coordinates: (3632) [...] }
  ▼ properties: Object { name: "Carrera de montaña a la hora del almuerzo",
    ► coordTimes: Array(3632) [ "2024-01-05T10:56:10Z", "2024-01-05T10:56:1
    ► heartRates: Array(3632) [ 82, 82, 81, ... ]
    name: "Carrera de montaña a la hora del almuerzo"
    time: "2024-01-05T10:56:10Z"
    type: "running"
    ► <prototype>: Object { ... }
    type: "Feature"
```

Nombre, tiempo de inicio y  
tipo de actividad

Esto es la hora en que se marcó  
cada punto y la frecuencia  
cardíaca

Ahora que ya comprendemos la estructura del JSON ya podemos trabajar con sus datos.

**Ejemplo:** vamos a obtener la frecuencia cardíaca media.

```
const hrArray = json.features[0].properties.heartRates;  
const averageHr = hrArray  
    .reduce( (acum, item) => acum+item, 0) / hrArray.length;  
console.log(averageHr);
```

Otra alternativa es trabajar directamente con los datos en XML, concretamente con el árbol DOM que generamos antes.

Al igual que hicimos antes, el primer pase será **comprender los datos con los que vamos a trabajar**.

```
index.html script.js pinos.gpx X
gpx_to_json > pinos.gpx
1 <?xml version="1.0" encoding="UTF-8"?>
2 <gpx creator="StravaGPX" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 <metadata>
4 <time>2023-12-31T08:38:00Z</time>
5 </metadata>
6 <trk>
7 <name>Por los pinos</name>
8 <type>running</type>
9 <trkseg>
10 <trkpt lat="42.5964210" lon="-5.5495650">
11 <ele>848.0</ele>
12 <time>2023-12-31T08:38:00Z</time>
13 <extensions>
14 <gpstpx:TrackPointExtension>
15 <gpstpx:hr>83</gpstpx:hr>
16 <gpstpx:cad>80</gpstpx:cad>
17 </gpstpx:TrackPointExtension>
18 </extensions>
19 </trkpt>
20 <trkpt lat="42.5964170" lon="-5.5495530">
21 <ele>848.0</ele>
22 <time>2023-12-31T08:38:01Z</time>
23 <extensions>
24 <gpstpx:TrackPointExtension>
25 <gpstpx:hr>83</gpstpx:hr>
26 <gpstpx:cad>80</gpstpx:cad>
27 </gpstpx:TrackPointExtension>
28 </extensions>
29 </trkpt>
30 <trkpt lat="42.5964140" lon="-5.5495410">
31 <ele>848.0</ele>
32 <time>2023-12-31T08:38:02Z</time>
33 <extensions>
```

Nombre, hora y tipo de actividad lo tengo en las etiquetas correspondientes.

Para cada punto registrado hay un nodo **trkpt** con la latitud y la longitud como atributos.

En ese nodo hay etiquetas para la elevación, tiempo y frecuencia cardíaca

¿Y cómo nos movemos por el árbol DOM del XML? Pues con las mismas funciones con las que nos movemos por el árbol DOM de una página web.

Vamos a ver como obtendríamos la frecuencia cardíaca media.

```
<trkpt lat="42.5964210" lon="-5.5495650">  
  <ele>848.0</ele>  
  <time>2023-12-31T08:38:00Z</time>  
  <extensions>  
    <gpstpx:TrackPointExtension>  
      <gpstpx:hr>83</gpstpx:hr>  
      <gpstpx:cad>80</gpstpx:cad>  
    </gpstpx:TrackPointExtension>  
  </extensions>  
</trkpt>
```

Recogemos todos los nodos con la etiqueta gpxtpx:hr

```
const trkpts = xmlDoc.getElementsByTagName("gpxtpx:hr")
```

Cada elemento del array que devuelve es un nodo del árbol DOM del XML

```
▼ HTMLCollection { 0: gpxtpx:hr , 1: gpxtpx:hr , 2: gpxtpx:hr , 8: gpxtpx:hr , 9: gpxtpx:hr , ... }  
  ▼ 0: <gpxtpx:hr>  
    ariaAtomic: null  
    ariaAutoComplete: null  
    ariaBusy: null  
    ariaChecked: null  
    ariaColCount: null  
    ariaColIndex: null  
    ariaColIndexText: null  
    ariaColSpan: null  
    .  
    --
```

El valor que nos interesa es el contenido de la etiqueta

```
<gpxtpx:TrackPointExtension>  
  <gpxtpx:hr>83</gpxtpx:hr>  
  <gpxtpx:cad>80</gpxtpx:cad>  
</gpxtpx:TrackPointExtension>
```

Vamos a ver si los extraemos

```
Array.from(trkpts).forEach(  
    item => console.log(item.textContent));
```

155

156

155

154

153

154

155




Así que vamos a calcular el valor medio como antes.

```
const suma = Array
    .from(trkpts)
    .reduce( (acum, item) =>
        acum+parseInt(item.textContent),
        0);
const averageHr = suma/Array.from(trkpts).length;
console.log(averageHr);
```

---

134.63711453744494

---

x\_to\_json >  script.js >  inputGPX.addEventListener('change') callback >  onload

```
const inputGPX = document.querySelector('#input-gpx');

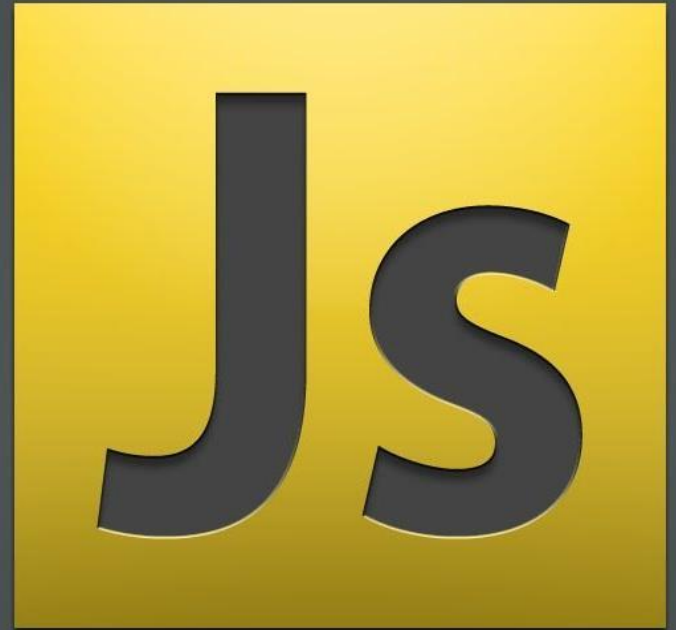
inputGPX.addEventListener( 'change' , () => {
  //almacenamos el archivo
  const file = inputGPX.files[0];
  console.log("file");
  console.log(file);
  if (file) {
    const reader = new FileReader();

    reader.onload = (event) => {
      const fileContent = event.target.result;
      const parser = new DOMParser();
      console.log("fileContent");
      const xmlDoc = parser.parseFromString(fileContent, 'application/xml');

      // Muestra el objeto xmlDoc en la consola
      console.log(xmlDoc);
      const json = toGeoJSON.gpx(xmlDoc);
      console.log(json);
    };
    const r = reader.readAsText(file);
    // console.log(json);
  }
});
```

9

FECHA Y HORA



El objeto **Date** almacena la fecha y hora, así como proporciona diversos métodos para administrarlas.

Para crear un nuevo objeto Date se instancia con **new Date()**. Se pueden pasar los siguientes parámetros:

- **new Date()**: almacenará la fecha y hora actual
- **new Date(*timestamp*)**: milisegundos desde 01/01/1970 (Unix Time). Admite valores negativos para fechas anteriores
- **new Date(datestring)**: cadena con la fecha que será analizada y convertida a fecha automáticamente
- **new Date(año, mes, día, hora, mins, segs, ms)**

Para acceder a los componentes de fecha tenemos las siguientes funciones.

- **getFullYear():** devuelve el año
- **getMonth():** devuelve el mes
- **getDate():** devuelve el día del mes
- **getHours(), getMinutes(), getSeconds(), getMilliseconds()**
- **getDay():** devuelve el día de la semana partiendo de 0 (domingo) hasta 6 (sábado)
- **getTime():** devuelve el timestamp (Unix time) en msecs

Análogamente, hay una serie de funciones para establecer los componentes:

- **setFullYear(year, [month], [date])**
- **setMonth(month, [date])**
- **setDate(date)**
- **setHours(hour, [min], [sec], [ms])**
- **setMinutes(min, [sec], [ms])**
- **setSeconds(sec, [ms])**
- **setMilliseconds(ms)**
- **setTime(timestamp)**

Hay que tener en cuenta que se aplica la autocorrección para los objetos Date cuando fijamos valores fuera de rango.

```
let date = new Date(2013, 1, 32); // ¿32 de Enero 2013?  
console.log(date); // 1 de Febrero de 2013
```

También se pueden establecer valores en 0 o negativos

```
date=new Date(2020, 0, 2); // 2 de enero de 2020  
date=new Date(2020, -4, 2); // 2 de septiembre de 2019
```

Si convertimos un objeto Date a número toma el valor del *timestamp* actual.

```
let now = new Date();  
console.log(+now);           //1675147090001
```

Esto implica que se pueden restar fechas y nos dará su diferencia en milisegundos.

```
let start = new Date(); // comienza a medir  
  
// la función hace su trabajo  
for (let i = 0; i < 100000; i++) {  
    let doSomething = i * i * i;  
}  
let end = new Date(); // termina de medir el tiempo  
console.log(`El tiempo transcurrido es de ${end - start} ms`);
```



Si solo queremos obtener el *timestamp* actual (como en el ejemplo anterior) podemos utilizar el método **Date.now()**.

Este método es equivalente a **new Date().getTime()** pero sin necesidad de crear una instancia del objeto, siendo por tanto más eficiente al no afectar a la recolección de basura.

```
console.log(Date.now()); // 1675147546300
```

# 10

USEO DE  
LIBRERÍAS:  
LEAFLET



Algo muy importante en cualquier lenguaje de programación es **no volver a programar aquello que ya ha programado otro.**

Este objetivo se consigue mediante las **librerías**, archivos de código que ya implementan funcionalidades que podemos utilizar en nuestros desarrollos web sin necesidad de volver a codificarlos.

Aunque pueden ser conceptos similares, no hay que confundir las **librerías** con los **frameworks**.

Una **librería** JavaScript es una pieza de código reutilizable que agrupa múltiples funciones/métodos/objetos generalmente con un fin común.

Ejemplo:

- **Leaflet**: manipulación de mapas
- **Lodash**: facilita el trabajo con arrays, objetos, strings, ...
- **jQuery**: manipulación del DOM, eventos, consultas AJAX, ...

Un **framework** es un conjunto de librerías que proporcionan código para realizar las tareas habituales de programación, pero proporcionando una estructura en torno al cual realizar el proyecto.

Un framework fuerza a realizar la aplicación siguiendo una determinada arquitectura.

Ejemplos:

- **Angular**
- **EmberJS**
- **VueJS**

Hay dos formas de incluir una librería en un documento HTML:

- Descargando el archivo js (disponible en la web de la propia librería) e insertándolo de la forma habitual mediante la etiqueta **<script>**
- Utilizando una **CDN** (*Content Delivery Network*), redes que realizan copias caché de archivos en servidores repartidos por todo el mundo para reducir tiempos de latencia.

Independientemente del método utilizado, deberá insertarse **antes** de nuestro script para tener disponibles las funciones cuando éste se ejecute.

La librería **Leaflet** proporciona una serie de funciones para insertar mapas interactivos en una página web.



an open-source JavaScript library  
for mobile-friendly interactive maps

[Overview](#) [Tutorials](#) [Docs](#) [Download](#) [Plugins](#) [Blog](#)

Si vamos al apartado de descargas podemos obtener el código fuente para guardarla directamente en nuestro servidor.

Nombre	Tamaño	Comprimido	Tipo	Modificado	CRC32
..			Carpeta de archivos		
images	6.503	6.503	Carpeta de archivos	18/04/2022 12:42	
leaflet.css	14.661	3.438	Documento de hoj...	18/04/2022 12:42	4936552F
leaflet.js	143.908	41.408	Archivo JavaScript	18/04/2022 12:43	D774C897
leaflet.js.map	196.318	71.620	Linker Address Map	18/04/2022 12:43	2C082D69
leaflet-src.esm.js	412.215	107.202	Archivo JavaScript	18/04/2022 12:43	9CEAFF6F
leaflet-src.esm.js.map	841.092	162.583	Linker Address Map	18/04/2022 12:43	AA1592B1
leaflet-src.js	437.177	109.082	Archivo JavaScript	18/04/2022 12:43	413C0D45
leaflet-src.js.map	841.184	163.763	Linker Address Map	18/04/2022 12:43	C03CF4BD

## Download Leaflet

Version	Description
<a href="#">Leaflet 1.9.3</a>	Stable version, released on November 18, 2022.
<a href="#">Leaflet 1.8.0</a>	Previous stable version, released on April 18, 2022.
<a href="#">Leaflet 2.0-dev</a>	In-progress version, developed on the main branch.



## leaflet.js

```

/* @preserve
 * Leaflet 1.8.0, a JS library for interactive maps. https://leafletjs.com
 * (c) 2010-2022 Vladimir Agafonkin, (c) 2010-2011 CloudMade
 */

!function(t,i){"object"==typeof exports&&"undefined"!=typeof module?i:
((t="undefined"!=typeof globalThis?globalThis:t||self).Leaflet=
length;n<o;n++)for(i in e=arguments[n])t[i]=e[i];return t}var R=Object.create||function(t){return N.prototype=t,new N};function N(){function a
(t,i){var e=Array.prototype.slice;if(t.bind)return t.bind.apply(t,e.call(arguments,1));var n=e.call(arguments,2);return function(){return t.
apply(i,n.length?n.concat(e.call(arguments)):arguments)}}var D=0;function h(t){return"leaflet_id"in t||(t._leaflet_id=++D),t._leaflet_id}
function j(t,i,e){var n,o,s=function(){n=!1,o&&(r.apply(e,o),o=!1)},r=function(){n?o=arguments:(t.apply(e,arguments),setTimeout(s,i),n=!0)};
return r}function H(t,i,e){var n=i[1],i=i[0],o=n-i;return t===n&&e?((t-i)%o+o)%i:function u(){return!1}function e(t,i){if(!1===i)return t;
i=Math.pow(10,void 0===i?6:i);return Math.round(t*i)/i}function W(t){return t.trim?t.trim():t.replace(/^\s+|\s+$/g,"")}function F(t){return W
(t).split(/\s+/)}function c(t,i){for(var e in Object.prototype.hasOwnProperty.call(t,"options"))|(t.options=t.options?R(t.options):{}),i)t.
options[e]=i[e];return t.options}function U(t,i,e){var n,o=[];for(n in t)o.push(encodeURIComponent(e?n.toUpperCase():n)+"="+encodeURIComponent(t
[n]));return i&&-1!==i.indexOf("?")?"&":"?"+"o.join("&");var V=/\{ *([\w_ -]+) *\}/g;function q(t,e){return t.replace(V,function(t,i){i=e[i];if
(void 0===i)throw new Error("No value provided for variable "+t);return i="function"==typeof i?i(e):i})}var d=Array.isArray||function(t){return
[object Array]===Object.prototype.toString.call(t)};function G(t,i){for(var e=0;e<t.length;e++)if(t[e]===i)return e;return-1}var K="data:image/
gif;base64,R0lGODlhAQABAAD/ACwAAAAAAQABAAACADs=";function Y(t){return window["webkit"+t]||window["moz"+t]||window["ms"+t]}var X=0;function J(t)

```

## leaflet-src.js

```

/* @preserve
 * Leaflet 1.8.0, a JS library for interactive maps. https://leafletjs.com
 * (c) 2010-2022 Vladimir Agafonkin, (c) 2010-2011 CloudMade
 */

(function (global, factory) {
  typeof exports === 'object' && typeof module !== 'undefined' ? factory(exports) :
  typeof define === 'function' && define.amd ? define(['exports'], factory) :
  (global = typeof globalThis !== 'undefined' ? globalThis : global || self, factory(global.leaflet = {}));
})(this, (function (exports) { 'use strict';

  var version = "1.8.0";

  /*
   * @namespace Util
   *
   * Various utility functions, used by Leaflet internally.
   */

  // @function extend(dest: Object, src?: Object): Object
  // Merges the properties of the `src` object (or multiple objects) into `dest` object and returns the latter.

```

La otra opción es utilizar un **CDN** para liberar de carga nuestro servidor y además obtener una menor latencia cuando el usuario se conecta desde una ubicación lejana.

### Using a Hosted Version of Leaflet

The latest stable Leaflet release is available on several CDN's — to start using it straight away, place this in the head of your HTML code:

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css" integrity="sha256-kLaT2GOSpl  
<script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js" integrity="sha256-WBkoXOWTeyKc10HuWtc+i2uENF|
```

Note that the [integrity hashes](#) are included for security when using Leaflet from CDN.

La otra opción es utilizar un **CDN** para liberar de carga nuestro servidor y además obtener una menor latencia cuando el usuario se conecta desde una ubicación lejana.

### Using a Hosted Version of Leaflet

The latest stable Leaflet release is available on several CDN's — to start using it straight away, place this in the head of your HTML code:

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css" integrity="sha256-kLaT2GOSpl  
<script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js" integrity="sha256-WBkoXOWTeyKc10HuWtc+i2uENF|
```

Note that the [integrity hashes](#) are included for security when using Leaflet from CDN.

El primer paso para insertar un mapa en nuestra página es **crear el espacio que lo va a contener.**

Este será un elemento `<div>` que deberá tener el **identificador map.**

Este `<div>` deberá tener definido un tamaño mediante CSS lo que determinará el tamaño del mapa.

```
<style>
  #map {
    width: 1280px;
    height: 640px;
  }
</style>

<div id="map"></div>
```

Ahora hay que crear el mapa.

Leaflet proporciona un objeto llamado **L** que tiene una serie de funciones para manipular el mapa. La mayoría de estas funciones devuelven el propio objeto mapa, **lo que permite encadenarlas**.

Ahora usaremos:

- **L.map('map')**: crea el objeto mapa y lo inserta en el elemento HTML con la etiqueta que se le pase como parámetro.
- **L.setView([lat, lon], zoom)**: establece la vista del mapa en las coordenadas indicadas con el nivel de zoom.

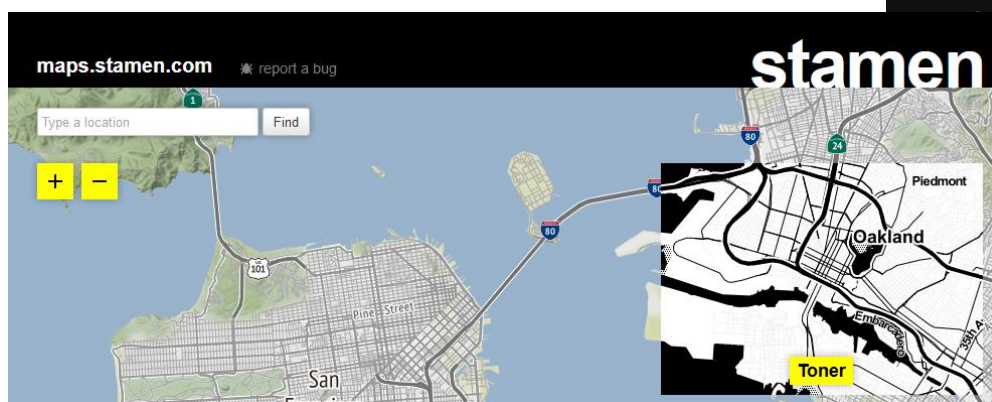
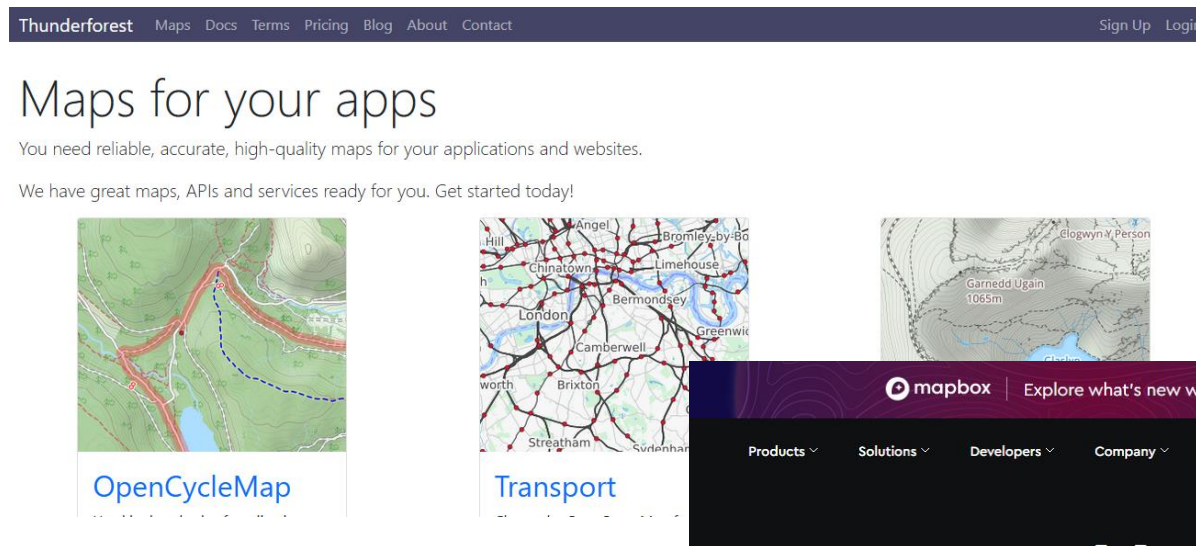
- **L.tileLayer(url, options).addTo(map)**: esta función indica de dónde se van a obtener los *tiles* del mapa. Por ejemplo, de OpenStreetMap. A continuación, lo añade al mapa.

```
let map = L.map('map')
    .setView([42.6000300, -5.570320], 13);
L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png',
{
    maxZoom: 19,
    attribution: '&copy; <a
href="http://www.openstreetmap.orgcopyright">OpenStreetMap</a
>'
}).addTo(map);
```

- **L.tileLayer(url, options).addTo(map)**: esta función indica de dónde se van a obtener los *tiles* del mapa. Por ejemplo, de OpenStreetMap. A continuación, lo añade al mapa.

```
let map = L.map('map').setView([42.6000300, -5.570320], 13);
L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png',
  {
    maxZoom: 19,
    attribution: '&copy; <a
href="http://www.openstreetmap.orgcopyright">OpenStreetMap</a
>'
  }).addTo(map);
```

Aparte de OpenStreetMap se pueden utilizar otros servicios de mapas online, como **Mapbox**, **Stamen** o **Thunderforest**



# Maps and location for developers

Precise location data and powerful developer tools to change the way we navigate the world.



## `L.marker([lat, lon], {options}).addTo(map)`

Añade un marcador al mapa.

Algunas de las propiedades de las opciones:

- **title**: *tooltip* que se mostrará al situar el cursor sobre él
- **opacity**: valor entre 0 y 1.0 que indica la opacidad del marcador.
- **riseOnHover**: animación al colocar el cursor sobre él
- **draggable**: si puede ser arrastrado

## `L.circle([lat, lon], {options}).addTo(map)`

Añade un círculo al mapa.

Algunas de las propiedades de las opciones:

- **color** y **fillColor**: cadena en formato CSS
- **fillOpacity**: opacidad
- **radius**: radio del círculo en metros
- **weight**: grosor del borde en píxeles

## Popups

Se puede añadir un popup a cualquiera de los elementos anteriores mediante la función **bindPopup()**. Este se mostrará al hacer click sobre el elemento o bien se mostrará abierto si añadimos la función **openPopup()**.

El parámetro que hay que pasarle a bindPopup es una cadena con código HTML que determinará el contenido del mismo.

```
var marker = L.marker([42.6000300, -5.570320], {  
  title: 'León',  
  opacity: 0.7,  
}).addTo(map);  
  
marker.bindPopup("<img src=./leon.jpg>").openPopup();
```



## **L.polyline(coords, options).addTo(map)**

Dibuja una polilínea que une las diferentes coordenadas que se le pasan como parámetro.

Las coordenadas se pasan como un array donde cada elemento es a su vez otro array con la latitud y la longitud del punto correspondiente.

Algunas opciones:

- **smoothFactor**: un mayor valor significa mejor rendimiento y una apariencia más suavizada, pero menor precisión.
- **color**
- **weight**

```
L.polyline([[42.70, -5.60], [42.800, -5.80], [42.750, -5.55], [42.7350, -5.5532] ], {  
  color: 'blue',  
  weight: 10,  
}).addTo(map);
```

