

JavaScript Avanzado



Agenda de hoy



- **Introducción al protocolo HTTP**

- Peticiones
- Respuestas
- Códigos de estado

- **Cientes HTTP**

- Postman
- Thunder Client

- **La tecnología AJAX**

- **WebStorage**

- Tipos de almacenamiento
- Local/Sessionstorage
 - Métodos



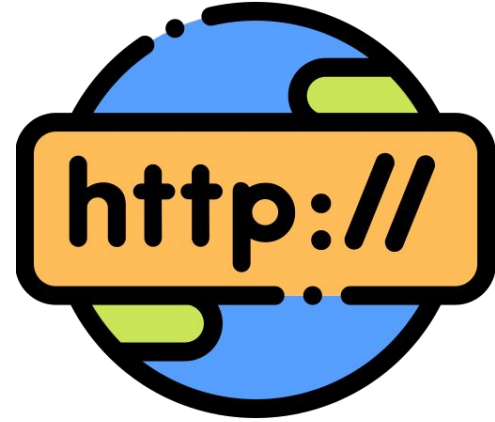
introducción a HTTP



introducción a HTTP

HyperText Transfer Protocol, es el nombre completo que le da vida a la sigla HTTP. Este protocolo es utilizado para intercambiar información a través de la red Internet.

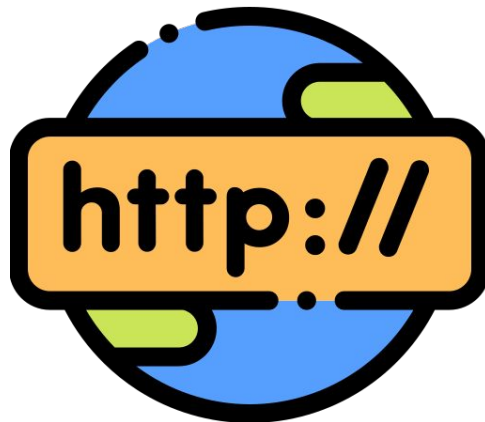
Este protocolo fue creado para transferencia de datos a través de una red utilizando el **modelo Cliente/Servidor**.



introducción a HTTP

En este modelo, el cliente realiza, (*envía*), una petición hacia el servidor, y este último le devuelve una respuesta asociada. En este modelo, el **Ciente** (*nosotros*) es quien inicia la comunicación, mientras que el **Servidor** se ocupa de responder.

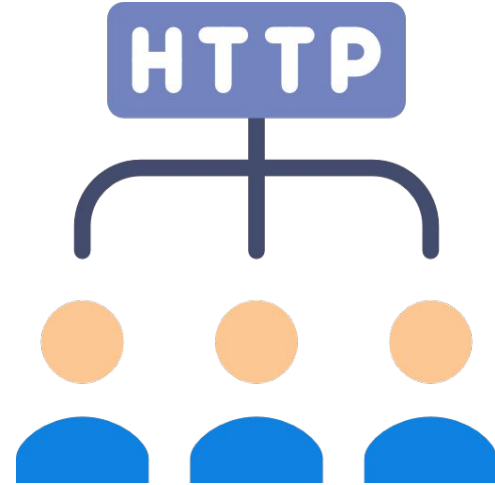
Este tipo de comunicación se realiza gracias a la tecnología conocida como **WebSockets**.



introducción a HTTP

El modelo de intercambio de datos que utiliza el protocolo HTTP, utiliza un protocolo sin estado.

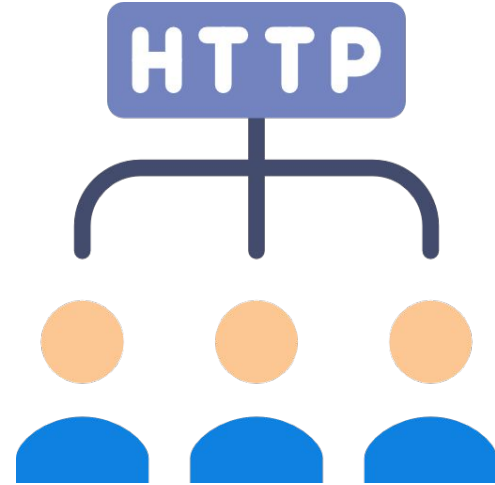
Esto hace alusión básicamente a que, en las reglas de este protocolo, toda la información está contenida en el intercambio de información.



introducción a HTTP

El protocolo en sí, no regula lo que sucede dentro del Servidor ni del lado del Cliente.

Solo se ocupa de establecer la comunicación entre las partes, regulando las entradas y salidas de datos entre ambos Agentes, pero no se ocupa del proceso interno de cada Agente.

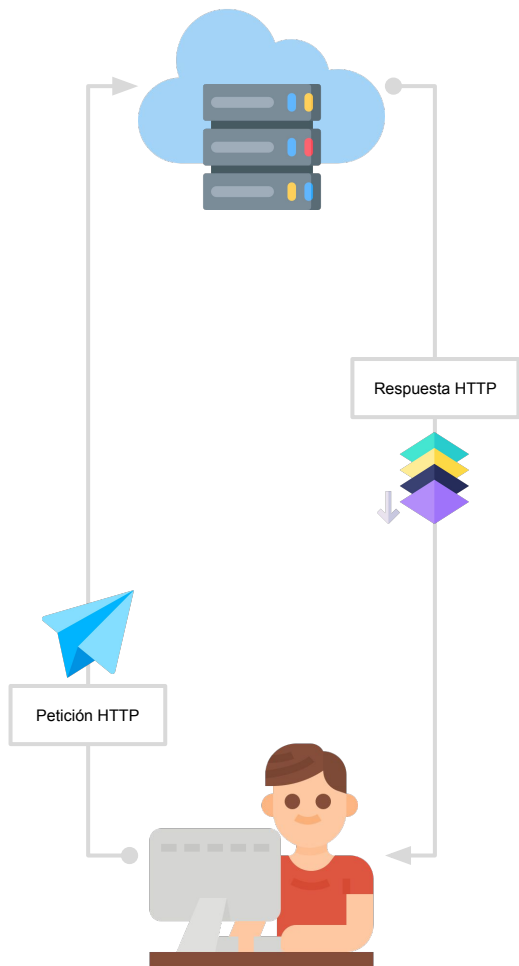


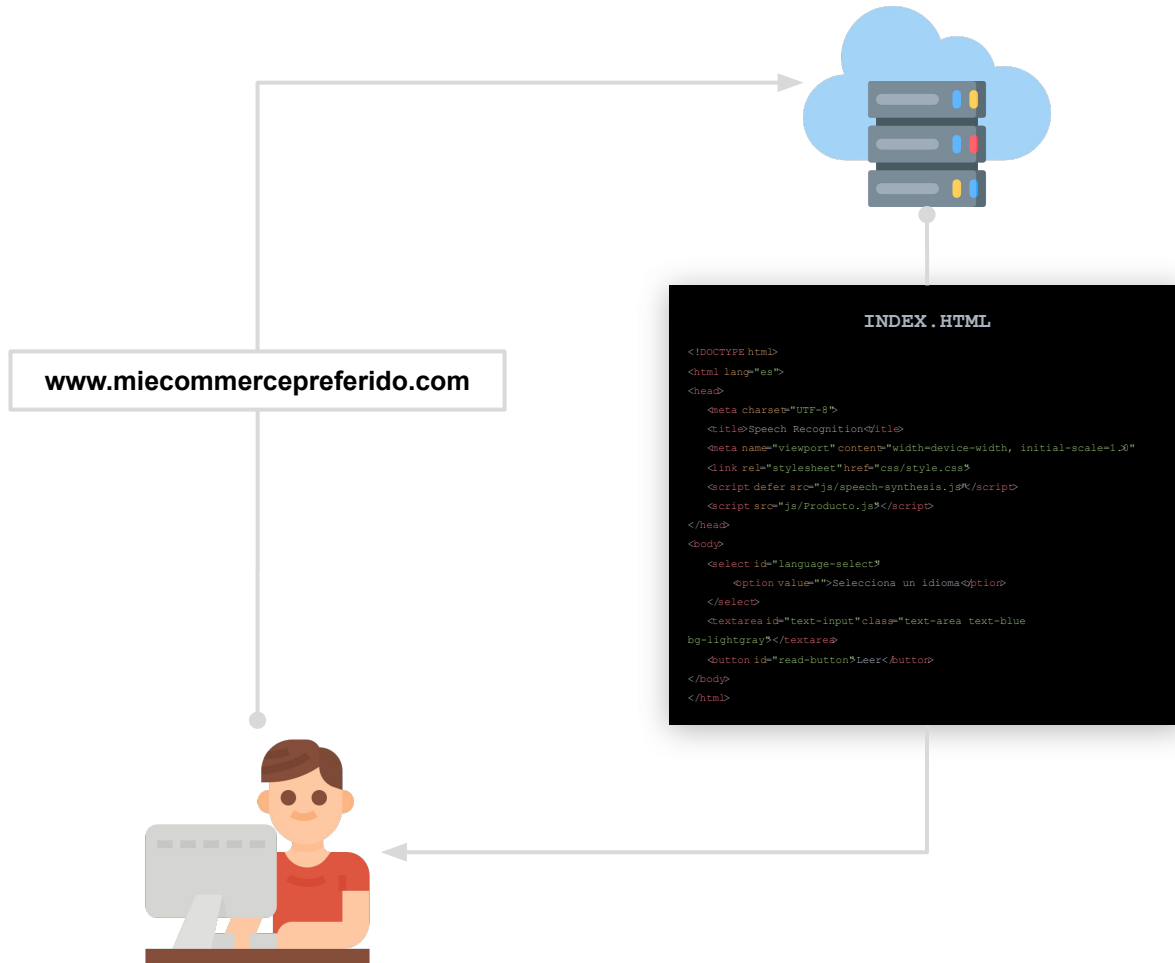
introducción a HTTP

Los mensajes que intercambiamos mediante el protocolo HTTP son, en esencia, textos. **HTTP sólo se ocupa de enviar y recibir texto formateado.**

Petición HTTP: mensaje enviado desde el cliente al servidor.

Respuesta HTTP: mensaje enviado desde el servidor al cliente.





Representación de la petición HTTP de un usuario a un sitio web.

Todo el contenido que viaja desde el servidor al usuario viaja en formato stream, o sea, texto plano.

introducción a HTTP

La información descargada vía stream, posee un encabezado y un cuerpo. En el encabezado (*header*), viaja información diversa, entre ella, el estado de la petición.

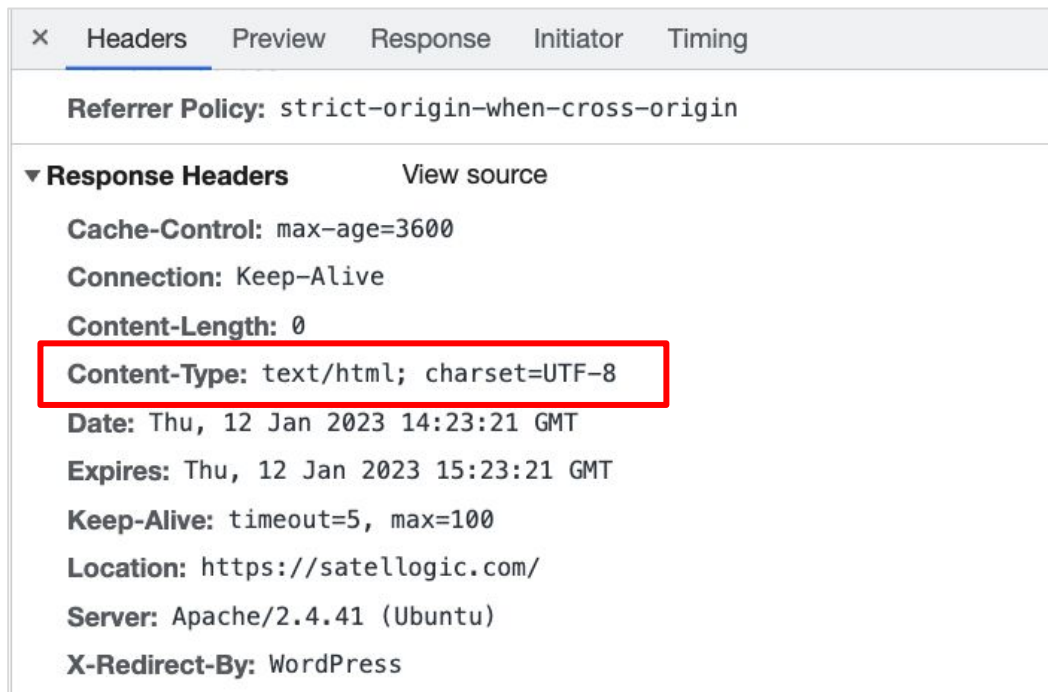
En el cuerpo (*body*), viaja la información referente al recurso que descargamos. En el caso del ejemplo anterior, todo el código del documento HTML en cuestión.



introducción a HTTP

Más allá de esto último, existen formas nativas de preguntar mediante JavaScript, si una web API está o no disponible en el navegador y/o en el sistema operativo donde está corriendo nuestra aplicación web.

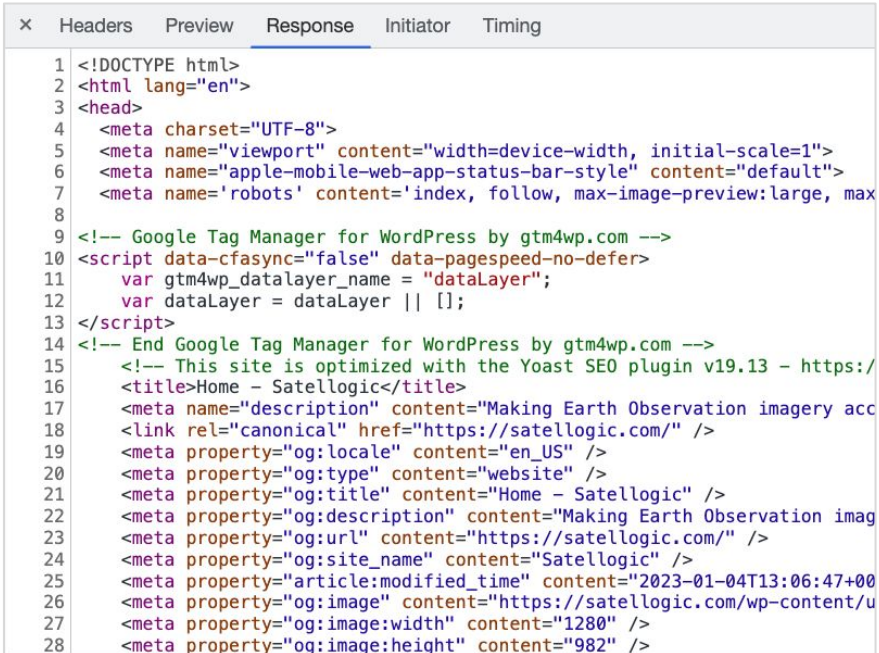
De esta forma, evitaremos que nuestra aplicación web falle, y hasta podremos buscar una alternativa de funcionamiento si la api web no está disponible.



introducción a HTTP

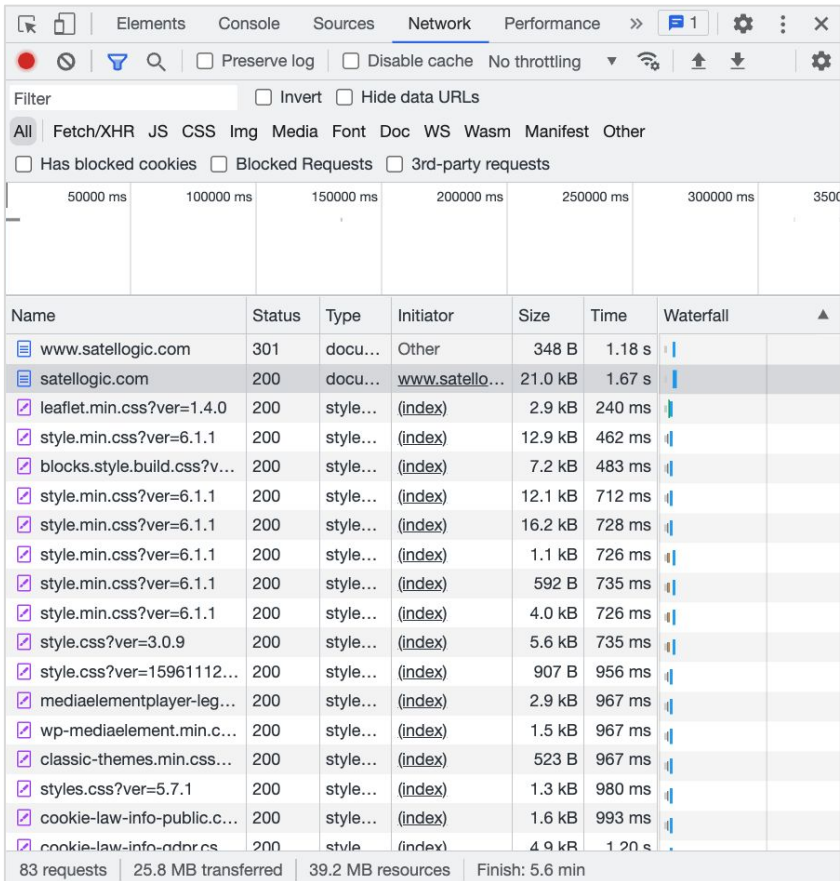
El cuerpo del mensaje, trae la información textual del contenido en cuestión.

En nuestro caso, el código del documento HTML de la página o documento principal.



```
x Headers Preview Response Initiator Timing
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <meta name="apple-mobile-web-app-status-bar-style" content="default">
7   <meta name="robots" content="index, follow, max-image-preview:large, max-
8
9 <!-- Google Tag Manager for WordPress by gtm4wp.com -->
10 <script data-cfasync="false" data-pagespeed-no-defer>
11   var gtm4wp_dataLayer_name = "dataLayer";
12   var dataLayer = dataLayer || [];
13 </script>
14 <!-- End Google Tag Manager for WordPress by gtm4wp.com -->
15 <!-- This site is optimized with the Yoast SEO plugin v19.13 - https://
16 <title>Home - Satellogic</title>
17 <meta name="description" content="Making Earth Observation imagery acc
18 <link rel="canonical" href="https://satellogic.com/" />
19 <meta property="og:locale" content="en_US" />
20 <meta property="og:type" content="website" />
21 <meta property="og:title" content="Home - Satellogic" />
22 <meta property="og:description" content="Making Earth Observation imag
23 <meta property="og:url" content="https://satellogic.com/" />
24 <meta property="og:site_name" content="Satellogic" />
25 <meta property="article:modified_time" content="2023-01-04T13:06:47+00
26 <meta property="og:image" content="https://satellogic.com/wp-content/u
27 <meta property="og:image:width" content="1280" />
28 <meta property="og:image:height" content="982" />
```

introducción a HTTP



The screenshot shows the Chrome DevTools Network tab. At the top, there are tabs for Elements, Console, Sources, Network (selected), and Performance. Below these are various filters and checkboxes: 'Filter' (empty), 'Invert', 'Hide data URLs', 'All' (selected), 'Fetch/XHR', 'JS', 'CSS', 'Img', 'Media', 'Font', 'Doc', 'WS', 'Wasm', 'Manifest', 'Other', 'Has blocked cookies', 'Blocked Requests', and '3rd-party requests'. A timeline at the top shows a sequence of requests from 0 to 350,000 ms. Below the timeline is a table of requests.

Name	Status	Type	Initiator	Size	Time	Waterfall	
www.satellogic.com	301	docu...	Other	348 B	1.18 s		
satellogic.com	200	docu...	www.satello...	21.0 kB	1.67 s		
leaflet.min.css?ver=1.4.0	200	style...	(index)	2.9 kB	240 ms		
style.min.css?ver=6.1.1	200	style...	(index)	12.9 kB	462 ms		
blocks.style.build.css?v...	200	style...	(index)	7.2 kB	483 ms		
style.min.css?ver=6.1.1	200	style...	(index)	12.1 kB	712 ms		
style.min.css?ver=6.1.1	200	style...	(index)	16.2 kB	728 ms		
style.min.css?ver=6.1.1	200	style...	(index)	1.1 kB	726 ms		
style.min.css?ver=6.1.1	200	style...	(index)	592 B	735 ms		
style.min.css?ver=6.1.1	200	style...	(index)	4.0 kB	726 ms		
style.css?ver=3.0.9	200	style...	(index)	5.6 kB	735 ms		
style.css?ver=15961112...	200	style...	(index)	907 B	956 ms		
mediaelementplayer-leg...	200	style...	(index)	2.9 kB	967 ms		
wp-mediaelement.min.c...	200	style...	(index)	1.5 kB	967 ms		
classic-themes.min.css...	200	style...	(index)	523 B	967 ms		
styles.css?ver=5.7.1	200	style...	(index)	1.3 kB	980 ms		
cookie-law-info-public.c...	200	style...	(index)	1.6 kB	993 ms		
cookie-law-info-gdpr.c...	200	style...	(index)	4.9 kB	1.20 s		

83 requests | 25.8 MB transferred | 39.2 MB resources | Finish: 5.6 min

DevTools > Network es la herramienta indispensable que nos ayuda a entender todo lo que se descarga mediante peticiones HTTP en nuestro navegador web. El documento HTML principal y sus recursos asociados.



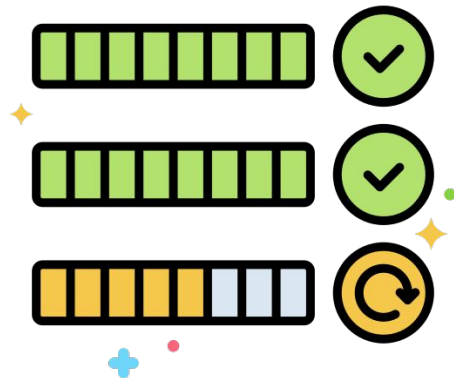
códigos de estado



códigos de estado

Con cada petición HTTP de respuesta viaja, dentro del encabezado, un código de estado. Estos son básicamente datos relacionados a la respuesta del servidor a nuestra petición.

Estos códigos son diferentes, y varían de acuerdo al tipo de petición. Se ocupan de confirmar si todo ha ido bien o surgió durante la descarga del recurso, algún error.



códigos de estado

Veamos a continuación, un resumen de los códigos de estado, de acuerdo al tipo de numeración y al significado de ésta:

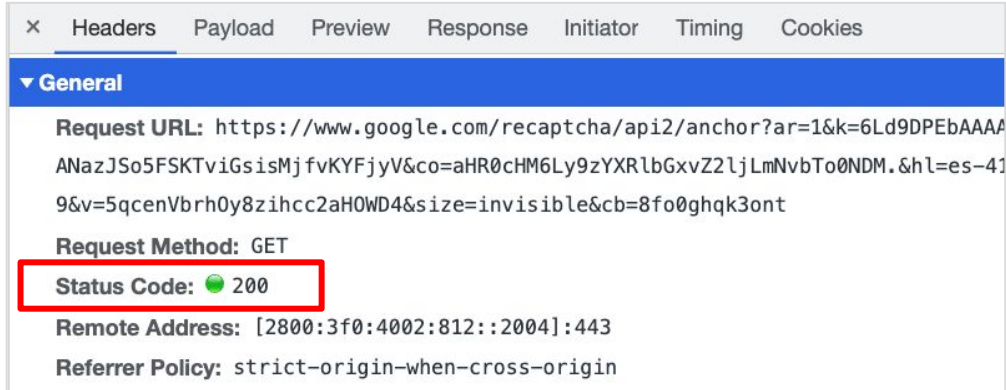
Código	Descripción
100	códigos informativos
200	códigos de éxito en la petición
300	códigos de redireccionamiento
400	códigos de error en el cliente
500	códigos de error en el servidor

Información complementaria, [aquí](#).



códigos de estado

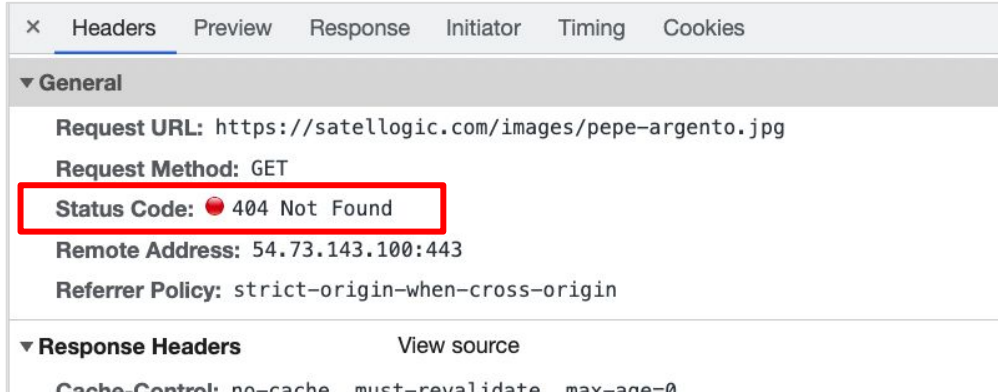
Dentro del rango de códigos de estado basados en **200** (201, 202, etc.), indica que la respuesta a la petición se ha realizado de manera exitosa.



códigos de estado

Dentro del rango de códigos de estado basados en **400** (401, 402, etc.), indica que ocurrió un error al intentar obtener un recurso remoto.

El más conocido de ellos: **el error 404**.



Clientes HTTP



Cientes HTTP

Un **cliente HTTP** es una aplicación que envía solicitudes a un servidor utilizando el **protocolo HTTP** (HyperText Transfer Protocol) y recibe respuestas de dicho servidor.

Es el "*iniciador*" de la comunicación en el modelo cliente-servidor.

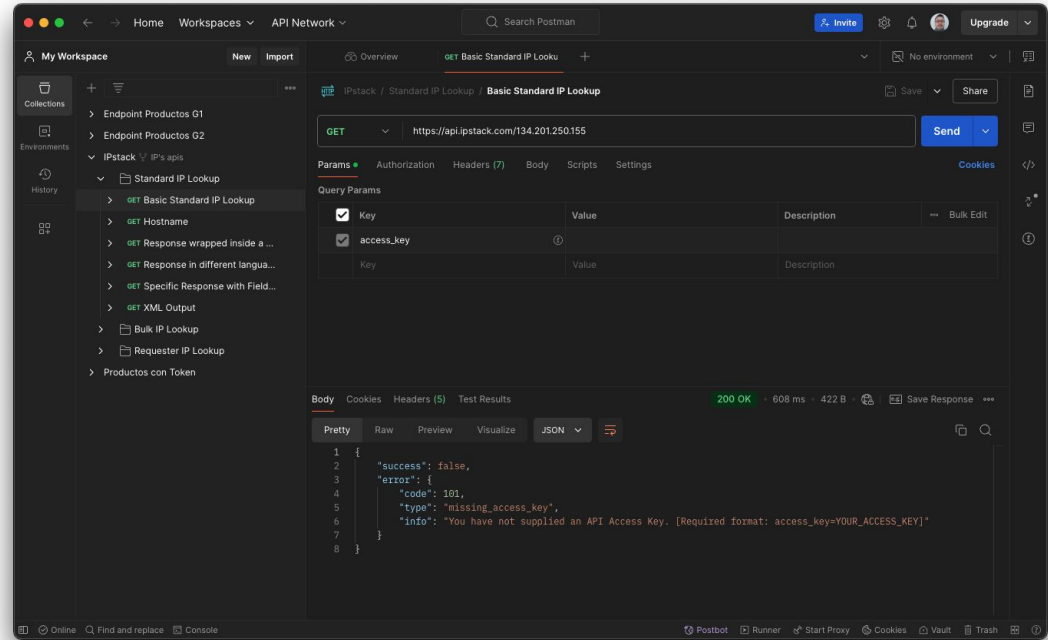


Cientes HTTP

POSTMAN

Es uno de los clientes HTTP más conocidos y utilizados desde hace al menos una década.

Tiene funcionalidades muy avanzadas que permiten no solo testear, sino aplicar una mínima programación en cada interacción, para validar datos adicionales.

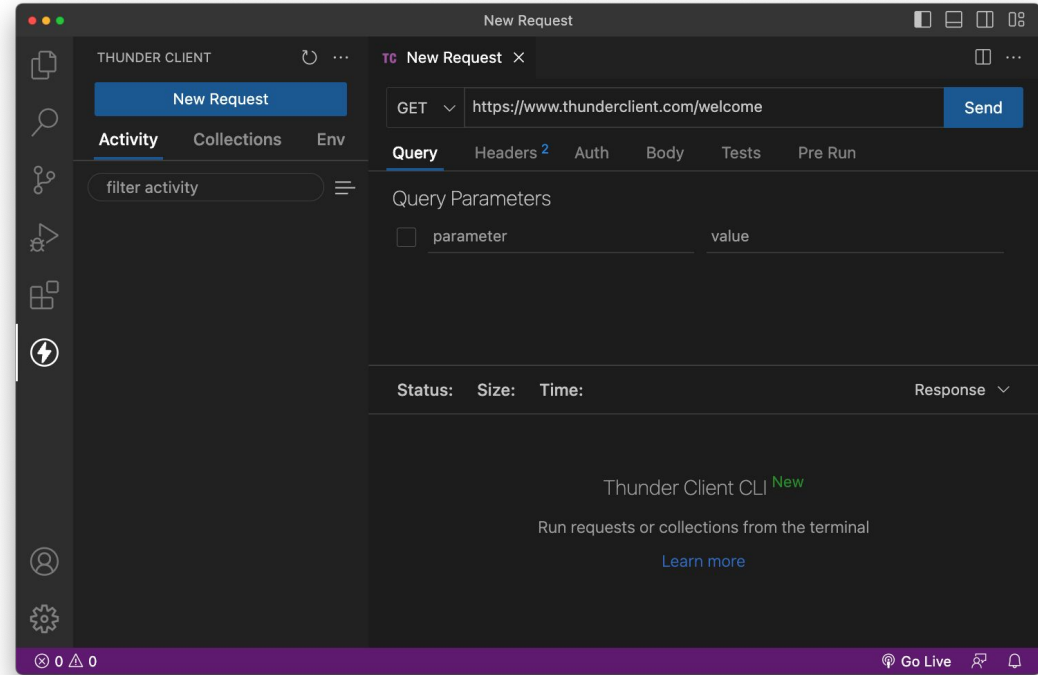


Cientes HTTP

THUNDER CLIENT

Es otro cliente HTTP, integrado en Visual Studio Code.

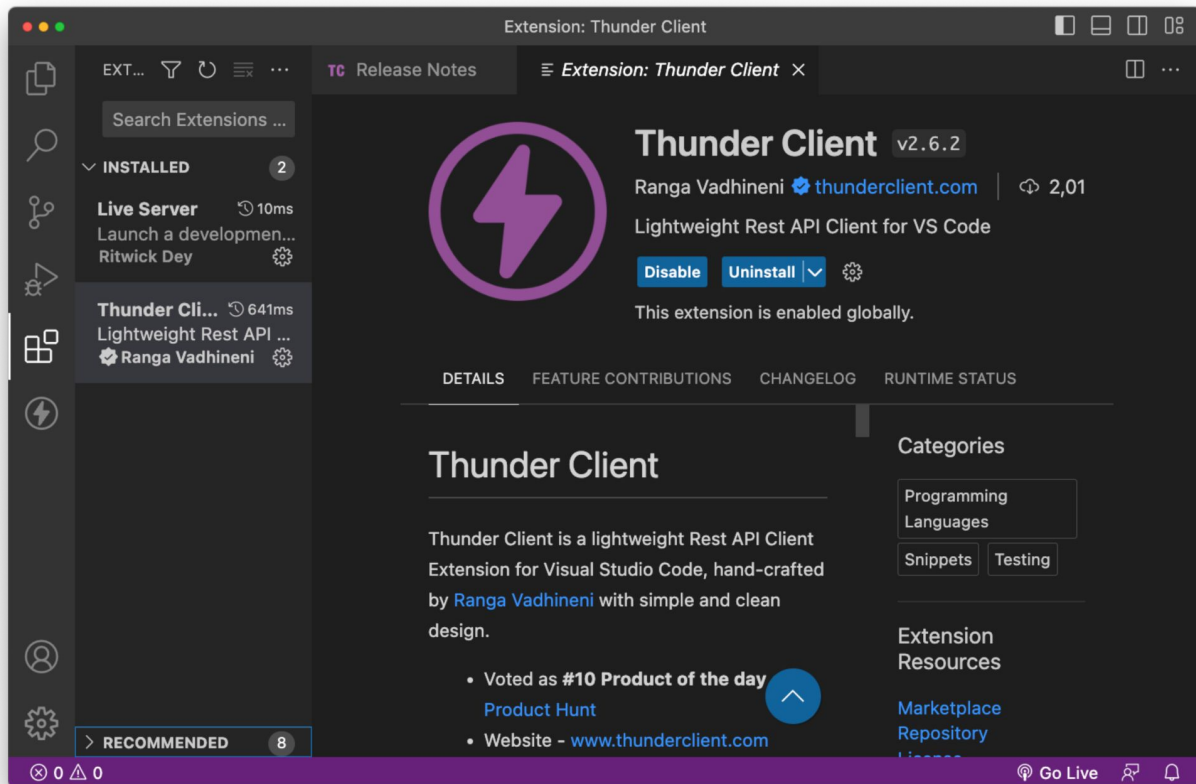
Se instala como extensión y se utiliza de igual forma que POSTMAN.



Interacción con un Servidor de Backend



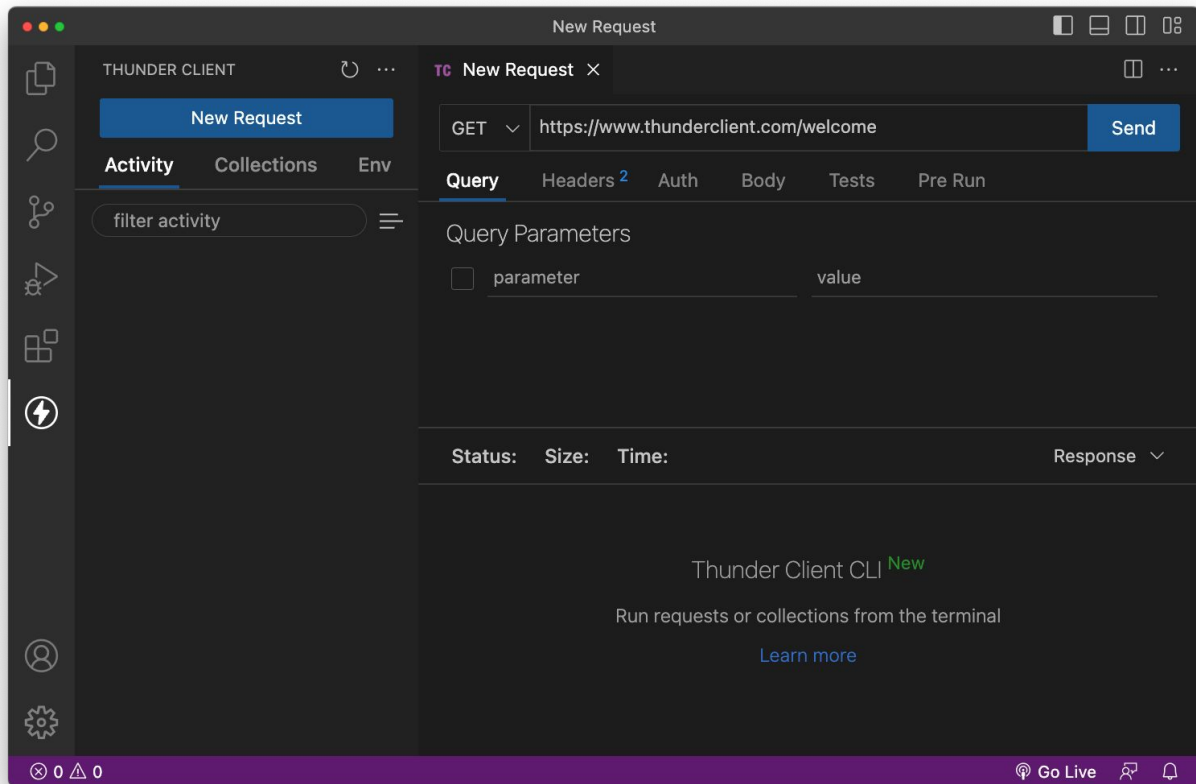
Interacción con un servidor de Backend



Instalaremos **Thunder Client** para peticionar datos a las aplicaciones Backend, a través de diferentes métodos que éste nos brinda.



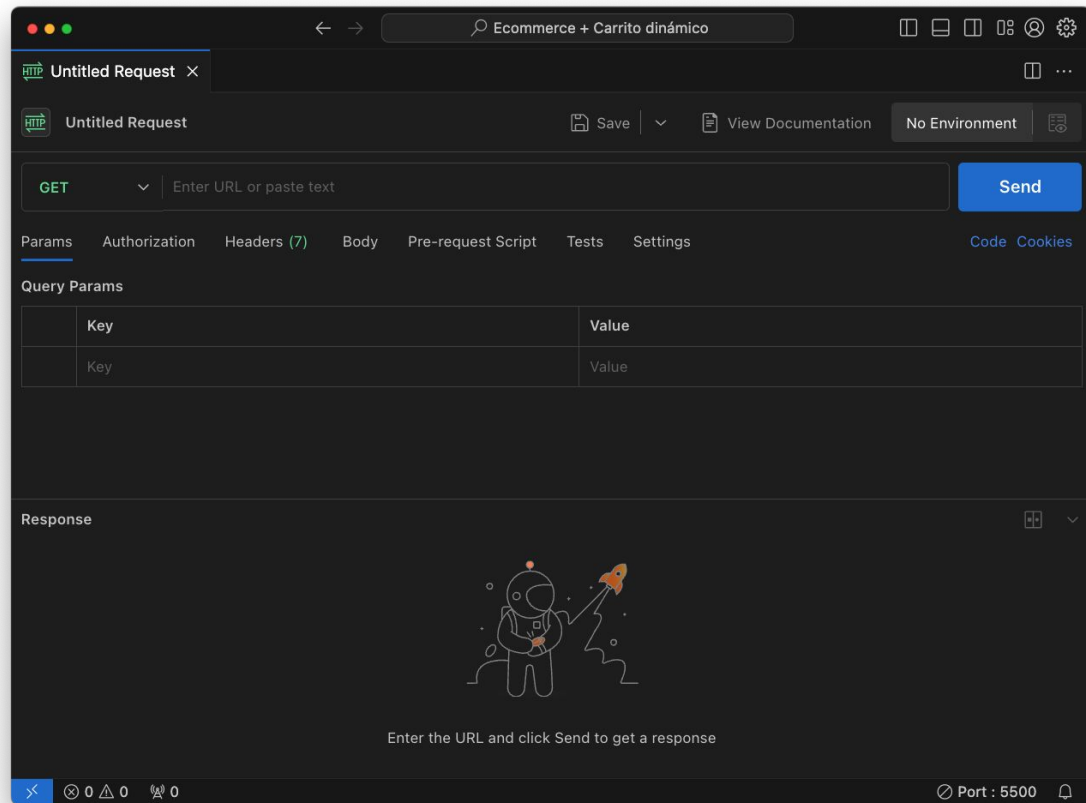
Interacción con un servidor de Backend



Su interfaz de uso es muy simple. A través de ella accederemos a los diferentes servicios web para peticionar por datos.



Interacción con un servidor de Backend

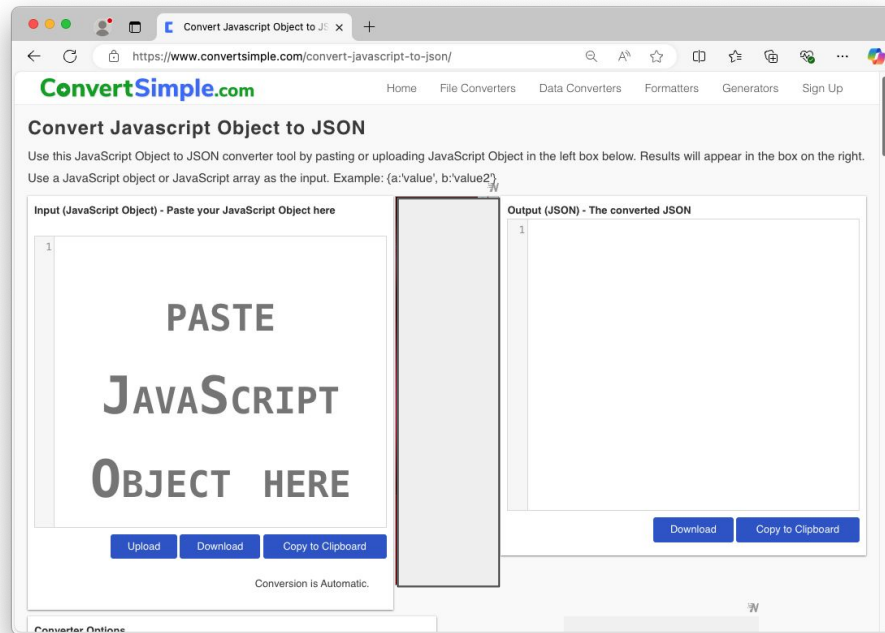


Si estás acostumbrado al cliente **POSTMAN**, puedes utilizarlo como alternativa, y hasta instalarlo como extensión en VS CODE.

También está disponible como extensión para VS Code.



Interacción con un servidor de Backend



Necesitaremos un conversor de objetos JS al formato JSON.

Podemos utilizar:

<https://www.convertsimple.com/convert-javascript-to-json/>





Espacio de prácticas





Espacio de prácticas



Veamos entre todos cómo utilizar una API REST de los servicios mencionados anteriormente. A través de sus diferentes posibilidades, podremos:

- Peticionar datos al servidor
- Definir filtros sobre los datos a mostrar
- Establecer el formato de respuesta
- Utilizar URL Params y Query Params
- Simular el alta, baja y modificación de datos (CRUD)

Utilizaremos para ello la API de [JSONPlaceholder](#) y de [RandomUser](#)



la tecnología AJAX



la tecnología AJAX

Hasta los primeros años de la década del 2000, la tecnología utilizada en Internet se basaba en el modelo **Server Side Rendering**.

Este modelo obligaba a que, por cada cambio que se producía en una página o aplicación web del lado del cliente, dicha información viajaba al servidor, este la procesaba, renderizaba la página con un resultado o respuesta, y devolvía la misma nuevamente al cliente.



la tecnología AJAX

Hoy en día, el modelo **Server Side Rendering** sigue vigente y tiene sus preferencias ante determinadas tecnologías aplicadas en el mundo web, antiguamente por cuestiones relacionadas a bajas velocidades de navegación y un bajo poder de procesamiento por parte de las computadoras, hacían bastante tedioso todo este proceso.



la tecnología AJAX

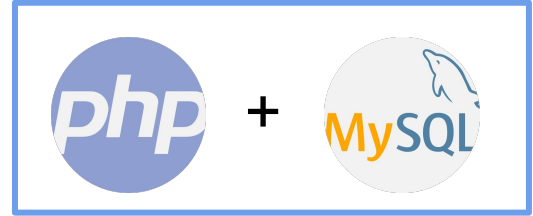
Tecnologías web como **PHP + MySQL**, **Java Server Pages**, o la misma **ASP.NET** de Microsoft, gestionaban este renderizado del lado del servidor, el cual hacía notorio un delay en los tiempos de respuesta para que el usuario vea contenido actualizado en un documento HTML.



la tecnología AJAX

Estas tecnologías en sí, funcionaban muy bien dentro de lo que era su mismo nicho tecnológico pero, pensar en cruzar información desde PHP hacia ASP.Net, o con Java Server Pages, era algo casi imposible de imaginar; al menos de una forma fácil y práctica.

Y no contamos el hecho de tener que formatear datos de cualquiera de estas plataformas para ser integrados a HTML de forma directa sin depender, por supuesto, de [SSR](#).

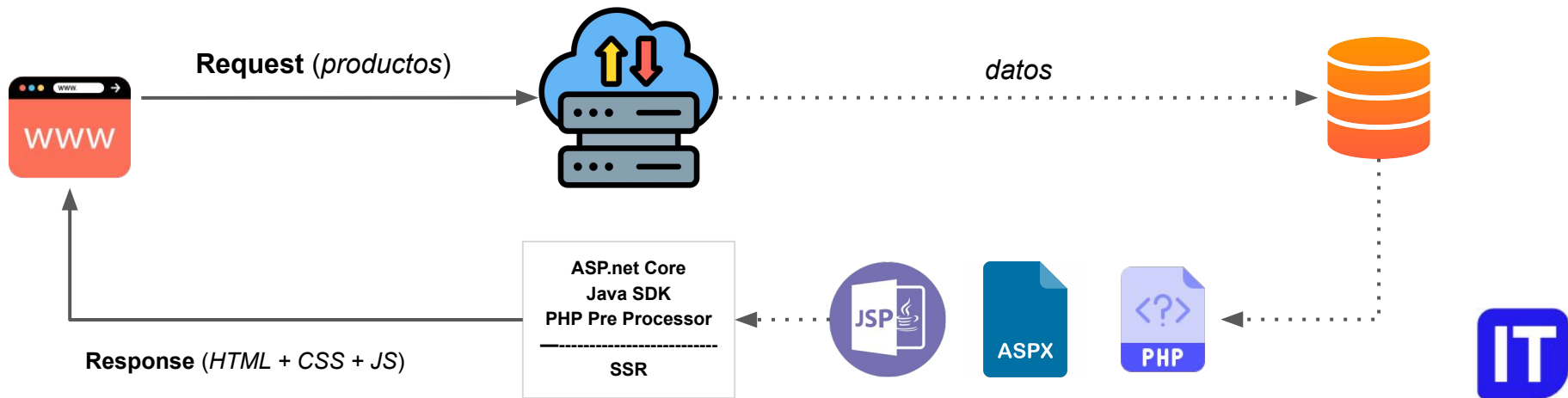


Sin tecnología AJAX

Representación gráfica de cómo funcionan tecnologías como .NET, Java o PHP en aplicaciones **frontend / backend**.

Todas necesitan de un pre-procesador o intérprete intermedio que pueda renderizar el código de la lógica de frontend + datos backend, para finalmente enviar lo que el navegador web puede interpretar: HTML+CSS+JS.

El servidor tiene doble tarea: SSR para el frontend en el lenguaje que este utilice + procesamiento y filtrado de datos del backend, para entremezclar el mismo con el frontend y que pueda descargarse todo al cliente (*browser*).



la tecnología AJAX

Para que cualquier tipo de tecnología frontend y backend pudiese intercambiar datos con cualquier otra tecnología opuesta, en el año 2002-2003, Microsoft ideó una propuesta original, denominada AJAX, basándose en el funcionamiento de las peticiones HTTP.



la tecnología AJAX

Asynchronous
JavaScript
And
XML

Esta consistía en utilizar una URL convencional para pedir información, (*datos*), a un servidor (*cualquiera sea este*), y que dicha información sea transferida mediante el mismo protocolo HTTP a la computadora cliente, en un formato estandarizado.



la tecnología AJAX

AJAX lograría así, no solo unificar información entre diferentes tecnologías de cliente y servidor, sino también aprovechar el intercambio de documentos web convencionales (*imágenes, HTML, CSS, XML, JavaScript, entre otros*), sin tener que recargar (*refresh - reload*) todo el documento HTML.



la tecnología AJAX

La **X**, de AJAX, hace referencia a la tecnología [XML](#) (*eXtensible Markup Language*). Esta nació en 1999 del estándar [SGML](#) (1986) y es similar, en estructura, al HTML que todos conocemos.

XML fue la propuesta de Microsoft para intercambiar datos entre Servidores y Clientes, dentro de su plataforma de desarrollo .NET.



la tecnología AJAX

Los datos, obtenidos de una base de datos, se formateaban en una estructura basada en XML.

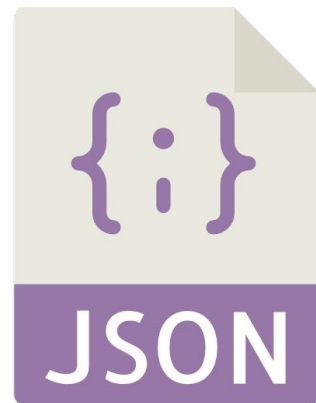
Luego, el XML se enviaba mediante una petición HTTP, hacia el cliente que había solicitado la información, para finalmente desestructurar la misma y mostrarla en pantalla.



la tecnología AJAX

Con el tiempo, nacería la tecnología JSON, la cual ganaría protagonismo por su simpleza y semejanza a una estructura de objetos JavaScript.

Así, JSON terminaría ganándole el terreno a la tecnología XML como un formato preferido para el intercambio de datos. Aún así, el término AJAX como tecnología, no fue cambiado.



estructura de datos XML

Status: 200 OK Size: 1.66 KB Time: 291 ms

Response Headers¹⁶ Cookies Results Docs

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <user>
3   <results>
4     <gender>female</gender>
5     <name>
6       <title>Miss</title>
7       <first>Troyana</first>
8       <last>Burko</last>
9     </name>
10    <location>
11      <street>
12        <number>6697</number>
13        <name>8-ma liniya</name>
14      </street>
15      <city>Kriviy Rig</city>
16      <state>Dnipropetrovska</state>
17      <country>Ukraine</country>
18      <postcode>41276</postcode>
19      <coordinates>
20        <latitude>78.5584</latitude>
21        <longitude>112.0579</longitude>
22      </coordinates>
23      <timezone>
24        <offset>+8:00</offset>
25        <description>Beijing, Perth, Singapore, Hong Kong</description>
26      </timezone>
27    </location>
28    <email>troyana.burko@example.com</email>
29    <login>
```

estructura de datos JSON

Status: 200 OK Size: 1.15 KB Time: 684 ms

Response Headers¹⁶ Cookies Results Docs

```
1 {
2   "results": [
3     {
4       "gender": "male",
5       "name": {
6         "title": "Mr",
7         "first": "Pascual",
8         "last": "Armas"
9       },
10      "location": {
11        "street": {
12          "number": 9459,
13          "name": "Corredor Vera"
14        },
15        "city": "Tlaltizapan",
16        "state": "Veracruz",
17        "country": "Mexico",
18        "postcode": 93596,
19        "coordinates": {
20          "latitude": "23.2536",
21          "longitude": "-51.3049"
22        },
23        "timezone": {
24          "offset": "+5:00",
25          "description": "Ekaterinburg, Islamabad, Karachi, Tashkent"
26        }
27      },

```

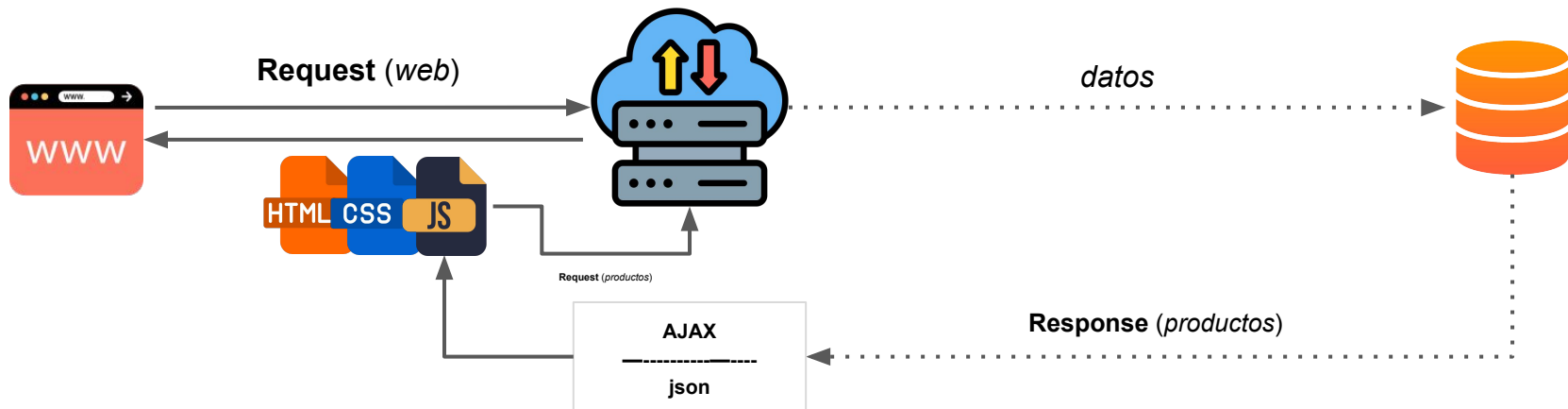


Con tecnología AJAX

La tecnología AJAX permite descargar el contenido HTML+CSS+JS del servidor, prácticamente sin SSR. Luego, JS es quien peticiona la estructura de datos (*productos, en nuestro ejemplo*). El servidor escucha la petición, dialoga con la bb.dd. y prepara la información en formato JSON.

Finalmente, solo se descarga la información en formato de texto, JS la transforma a una estructura de datos JSON (*objeto literal || array de objetos*), y la representa en el documento HTML de forma dinámica.

El cliente (*browser*) es quien hace este último trabajo, quitándole carga al servidor web.





la tecnología AJAX



Esto es un ejemplo simple de cómo esta tecnología permite intercambiar información entre una tecnología cliente determinada, y otra tecnología de servidor, sin importar qué tecnología hay de cada lado.

Para entender mejor esto; podemos tener una aplicación de escritorio, una aplicación móvil, o una aplicación web (*cliente*) peticionando datos y, del otro extremo, cualquier tipo de tecnología mencionada anteriormente {*Java Server Pages, PHP, ASP.Net, o alguna más nueva como ser GoLang, Node JS, Python Django*} (*servidor*).



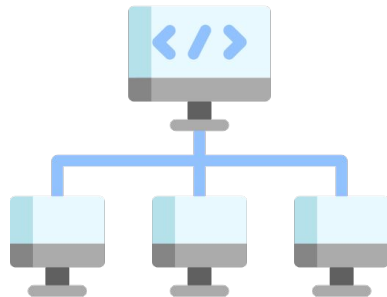
Cómo accedemos a datos en servidores



Cómo accedemos a datos en servidores

Describamos a continuación a todos los actores que tienen un papel para acceder a datos almacenados en servidores (*web, en nuestro caso*). Por supuesto que, en primera instancia, debe existir un “*Cliente*” que quiera peticionar dichos datos.

Este cliente, como bien dijimos, puede ser una WebApp, Mobile App, Native App, IoT Device, Smart TV, o cualquier otro dispositivo que tenga capacidades de mostrar datos visualmente.



Cómo accedemos a datos en servidores

Del lado del servidor debe existir: un servidor web, una aplicación web, una base (o almacén) de datos, algún mecanismo de validación de usuario (*Login, Token, validación por URL, etcétera*).

Cliente



Protocolo



Servidor



web server



server App /
Middleware



security



database
/
datastorage



URL



Cómo accedemos a datos en servidores



Este será el protocolo utilizado para llevar y traer las peticiones desde el cliente al servidor, y viceversa.



Una URL será el formato para realizar peticiones al servidor, y que este nos devuelva la información solicitada.

***JSON** o **XML**, son el formato de transporte de los datos en sí.*



documentación, ¡siempre!



Y para poder conjugar todo esto, tener documentación desarrollada para entender cómo pedir los datos y qué recibiremos como respuesta, es algo clave.

Esta documentación nos sirve a nosotros, como desarrolladores, como también a cualquier otro equipo o persona técnica que necesite acceder a la misma información.

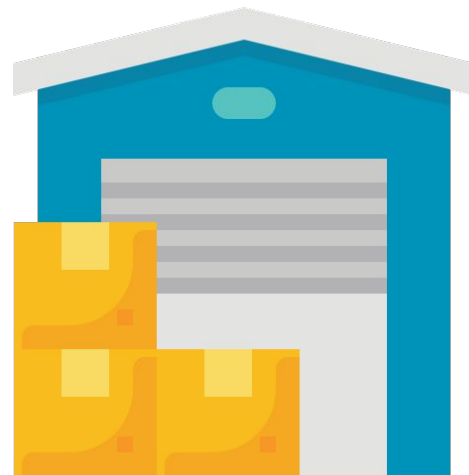


WebStorage

WebStorage

Dentro de la propuesta de construcción del lenguaje HTML5, existió un apartado con la definición de web storage.








En este se incluyó un set de diferentes propuestas para almacenar información del lado del usuario, de manera más efectiva a lo que proponen las **Cookies**, existentes en web browsers desde el nacimiento de estos.



WebStorage

En este set de opciones, alojado dentro de **DevTools > Application > Storage**, encontramos las diferentes opciones disponibles hoy en los navegadores web.

Storage

- ▶  Local Storage
- ▶  Session Storage
-  IndexedDB
-  Web SQL
- ▶  Cookies
-  Trust Tokens
-  Interest Groups








Cache

-  Cache Storage
-  Back/forward cache

WebStorage

- Funciona en modo SandBox
- Posee una capacidad de almacenamiento del lado del usuario muy alta (hasta el 80% del espacio en disco)
- Su estructura es similar a las bases de datos del tipo Nosql
- Su curva de aprendizaje es algo compleja (*promesas JS + asincronismo*) { se recomienda [operarla mediante alguna librería JS](#) }
- Es la base del almacenamiento local de plataformas Premium (*youtube, spotify, netflix, etcétera*)

Storage

- ▶  Local Storage
- ▶  Session Storage
-  IndexedDB
-  Web SQL
- ▶  Cookies
-  Trust Tokens
-  Interest Groups








Cache

-  Cache Storage
-  Back/forward cache

WebStorage

- Grandiosa idea... *camino a desaparecer*
- Safari ya desactivó la característica
- Chrome/Edge ~~están en camino a hacer lo mismo~~ también la han desactivado
- Firefox nunca incluyó el soporte a Web SQL
- No posee mantenimiento desde 2011

Storage

- ▶  Local Storage
- ▶  Session Storage
-  IndexedDB
-  **Web SQL**
- ▶  Cookies
-  Trust Tokens
-  Interest Groups








Cache

-  Cache Storage
-  Back/forward cache

WebStorage

- Funcionan en modo SandBox
 - Almacenan hasta 10 MB del lado del usuario
 - Trabajan bajo el modelo clave-valor
 - Se pueden trabajar como un objeto JS
-
- **LocalStorage** persiste la información
 - **SessionStorage** almacena información hasta tanto se cierre la pestaña de navegación

Storage

- ▶  Local Storage
- ▶  Session Storage
-  IndexedDB
-  Web SQL
- ▶  Cookies
-  Trust Tokens
-  Interest Groups



Cache

-  Cache Storage
-  Back/forward cache

WebStorage

Local/Session (storage) son las que utilizaremos de forma frecuente.

Storage

- ▶  Local Storage
- ▶  Session Storage
-  IndexedDB
-  Web SQL
- ▶  Local Storage
- ▶  Session Storage
-  Interest Groups

Cache

-  Cache Storage
-  Back/forward cache

WebStorage

Dispone de una propiedad y varios métodos que nos permiten interactuar rápidamente con su contenido.

Método - Propiedad	Descripción
<code>.length</code> (p)	devuelve el número de ítems almacenados
<code>setItem(clave, valor)</code>	crea un nuevo ítem. clave = nombre, valor = dato a almacenar
<code>getItem(clave)</code>	recupera un ítem almacenado, a través de su clave
<code>removeItem(clave)</code>	elimina un ítem almacenado, a través de su clave

WebStorage

Adicionalmente, cuenta con dos métodos adicionales.

Método	Descripción
<code>clear()</code>	elimina todos los ítems existentes en Storage
<code>key(índice)</code>	obtiene la clave de una posición determinada del storage

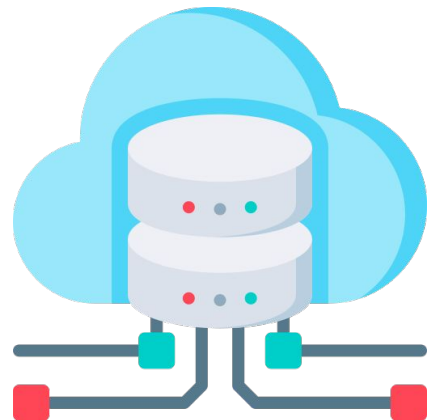
(*) Tanto el método `.clear()` como el método `.removeItem(clave)`, eliminan datos sin una validación de por medio. Tengamos cuidado al aplicar los mismos en el código de una aplicación web.



WebStorage

El tipo de almacenamiento de información bajo estos mecanismos de Storage, se debe realizar siempre en formato **String**. Solo podemos almacenar datos numéricos, booleanos, o cadenas de texto.

No es posible almacenar **objetos** o **arrays**, tal como están estructurados.



setItem() - getItem()

```
localStorage.setItem("demo", "Esto es una prueba del LocalStorage");  
localStorage.setItem("numero", 2);  
localStorage.setItem("boolean", false);  
  
alert(localStorage.getItem("demo"))  
alert(localStorage.getItem("numero"))  
alert(localStorage.getItem("boolean"))
```

setItem() - getItem()

<div>▼ Local Storage</div> <div>https://www.istea.edu.ar</div> <div>https://vars.hotjar.com</div> <div>https://www.google.com</div> <div>► Session Storage</div> <div>IndexedDB</div>	<div>⌂ Filter</div> <table><thead><tr><th>Key</th><th>Value</th></tr></thead><tbody><tr><td>Prueba</td><td>Esto es un texto de prueba almacenado en LocalStorage.</td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></tbody></table>	Key	Value	Prueba	Esto es un texto de prueba almacenado en LocalStorage.				
Key	Value								
Prueba	Esto es un texto de prueba almacenado en LocalStorage.								

removeItem()

The screenshot displays the Chrome DevTools interface. On the left, the 'Local Storage' section is expanded, showing a list of URLs: `https://www.istea.edu.ar`, `https://www.google.com`, and `https://vars.hotjar.com`. The 'Console' tab is selected at the bottom, showing 'No messages'. The 'Issues' panel is also visible, indicating '1 Issue'.

The 'Local Storage' table is shown with the following data:

Key	Value
Prueba	Esto es un texto de prueba almacenado en LocalStorage.

The 'Console' panel shows 'No messages'.

key() - length

```
// Acceso tipo Objeto
console.log(localStorage.demo);
console.log(localStorage.key(2));

// Iterar sobre las claves
for(let i=0; i<localStorage.length; i++) {
  let key = localStorage.key(i);
  alert(`${key}: ${localStorage.getItem(key)}`);
}
```

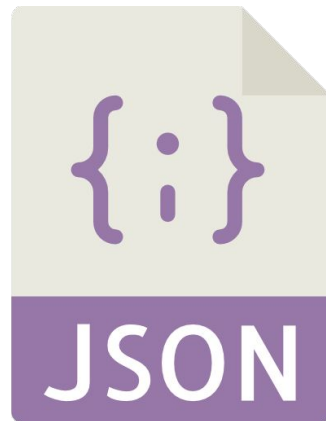
JSON



¿Qué es JSON?

JSON (*JavaScript Object Notation*) es un formato general para representar valores y objetos. No es más que un formato ligero de datos, con una estructura (notación) específica, que es totalmente compatible de forma nativa con Javascript.

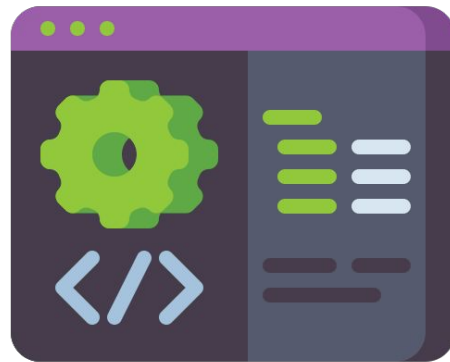
Como su nombre lo indica, JSON se basa en la sintaxis que tiene Javascript para crear objetos. Es comúnmente utilizado para enviar y almacenar datos en aplicaciones web.



¿Qué es JSON?

Con el tiempo, se convirtió en el formato de transporte de datos, no solo en JavaScript, sino también en el resto de los lenguajes de programación que deben “*dialogar*” con algún servidor proveedor de datos (*aplicaciones backend*).

Originalmente, [el formato XML](#) era el que predominaba en este terreno.



```

Users > fernando > Downloads > prueba.xml
1  <user>
2      <results>
3          <gender>male</gender>
4          <name>
5              <title>Mr</title>
6              <first>Lenni</first>
7              <last>Lehto</last>
8          </name>
9          <location>
10             <street>
11                 <number>9748</number>
12                 <name>Rautatiekatu</name>
13             </street>
14             <city>Säskylä</city>
15             <state>Central Finland</state>
16             <country>Finland</country>
17             <postcode>65975</postcode>
18             <coordinates>
19                 <latitude>-22.9944</latitude>
20                 <longitude>-29.5249</longitude>
21             </coordinates>
22             <timezone>
23                 <offset>+7:00</offset>
24                 <description>Bangkok, Hanoi, Jakarta</description>
25             </timezone>
26         </location>
27         <email>lenni.lehto@example.com</email>
28         <login>
29             <uuid>87dfad09-7f2c-462c-85a0-a7177668e0ef</uuid>
30             <username>redbird681</username>
31             <password>fallen</password>
32             <salt>D7lXajWq</salt>
33             <md5>eabab72553a75a52877bb32140b63dce</md5>
34             <sha1>ded6690dbc7f42c7d4d0a0f37419818e6d54eb89</sha1>
35             <sha256>2ebfab9348b9e162a494311dc4d68b6e87585d6d5e73af
36                 4927948ac93c1561d7</sha256>
37         </login>
38         <dob>
39             <date>1972-09-01T18:12:11.961Z</date>
40             <age>50</age>

```

```

Users > fernando > Downloads > prueba.json > ...
1  {
2      "results": [
3          {
4              "gender": "female",
5              "name": {
6                  "title": "Ms",
7                  "first": "Alice",
8                  "last": "Ellis"
9              },
10             "location": {
11                 "street": {
12                     "number": 904,
13                     "name": "Manor Road"
14                 },
15                 "city": "Salford",
16                 "state": "Hertfordshire",
17                 "country": "United Kingdom",
18                 "postcode": "HG00 9LF",
19                 "coordinates": {
20                     "latitude": "-7.3168",
21                     "longitude": "5.7604"
22                 },
23                 "timezone": {
24                     "offset": "-11:00",
25                     "description": "Midway Island, Samoa"
26                 }
27             },
28             "email": "alice.ellis@example.com",
29             "login": {
30                 "uuid": "03b016b7-1b7b-4606-8e5a-57ec910a75a7",
31                 "username": "redfrog496",
32                 "password": "fortuna",
33                 "salt": "RCSxbAa6",
34                 "md5": "8918fc3b2d87f033da6badd53c76fcbe",
35                 "sha1":
36                     "0da48bbd6927c76a1a1023ca0f486d1841d768a6",
37                 "sha256":
38                     "594ff0bb14df8d4034e8e5e684e7afb508bccc41d44f568
39                     62037cbe1df099b1f"
40             }
41         }
42     ]
43 }

```



JavaScript JSON

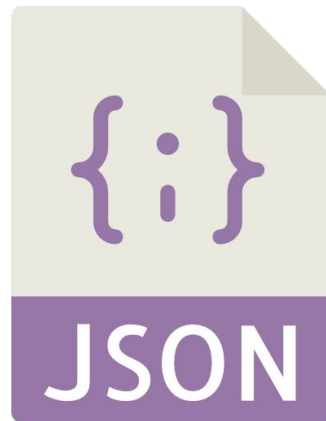
(objeto global)



JavaScript JSON

JavaScript cuenta con un objeto nativo denominado **JSON**.

El mismo posee dos métodos, los cuales permiten interactuar entre un array de objetos literales o el contenido en formato JSON, recibido desde algún servicio proveedor de datos (*o generado por nuestra aplicación web*).



JavaScript JSON

Método	Descripción
<code>.parse()</code>	Convierte en un objeto o array de objetos Javascript, el contenido recibido en formato de transporte de datos.
<code>.stringify()</code>	recibe un objeto literal o array de objetos literales, y lo convierte en un formato de transporte, del tipo String.

JavaScript JSON

Array de objetos en JS

```
const productos = [{id: 1, nombre: 'Naranjas umbligo', precio: 195.00},  
                   {id: 2, nombre: 'Mandarinas dulces', precio: 275.00},  
                   {id: 3, nombre: 'Pomelos rosados', precio: 165.00},  
                   {id: 4, nombre: 'Limonos 4 estaciones', precio: 245.00}]
```

Convierto un array de objetos y lo guardo en LS

```
let prodsJSON = JSON.stringify(productos);  
localStorage.setItem('Productos', prodsJSON);
```

Convertimos un objeto JS a string, luego lo almacenamos en LocalStorage.



JavaScript JSON



Recupero array desde LS y lo convierto en objetos JS

```
const productosLS = localStorage.getItem('Productos');  
  
productos = JSON.parse(productosLS);
```

Aplicamos la inversa: recuperamos un elemento de LocalStorage, el cual sabemos que es un array de objetos (*convertido a String*). Luego lo pasamos por el método **parse()**, y se convierte nuevamente en un array de objetos JS.

¡Gracias!

educación IT