



# HTTP



## v20240916

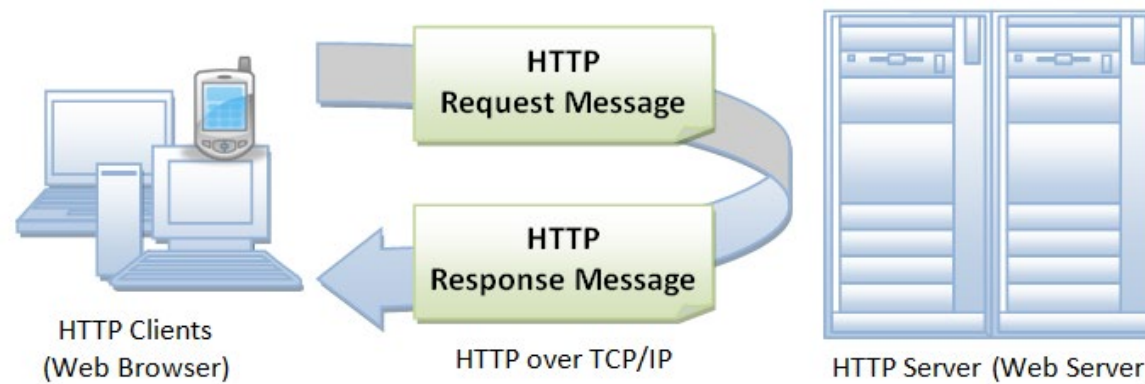
- David González Márquez

# Introducción

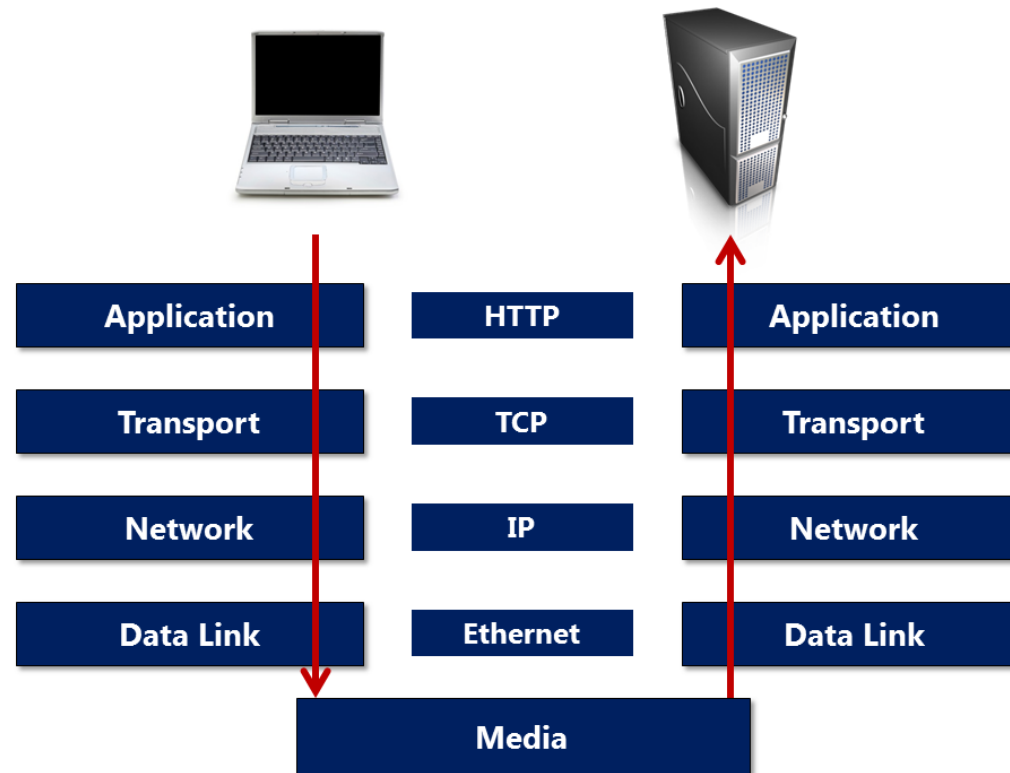
- Hypertext Transfer Protocol (HTTP)
  - Protocolo de Transferencia de Hipertexto
- Capa de aplicación
- Transmitir documentos hipermedia como HTML
- Comunicación entre clientes y servidores
  - Se puede usar con otros propósitos
- Protocolo de transacciones (petición – respuesta)
- Protocolo sin estado (stateless)
- Se suele usar con TCP/IP
  - Se puede usar con cualquier capa de transporte fiable (pej. RUDP)

# Introducción

- El cliente abre una conexión (pej. TCP)
- El cliente inicia una petición HTTP (request)
- El servidor devuelve la respuesta HTTP (response)
- Se cierra la conexión



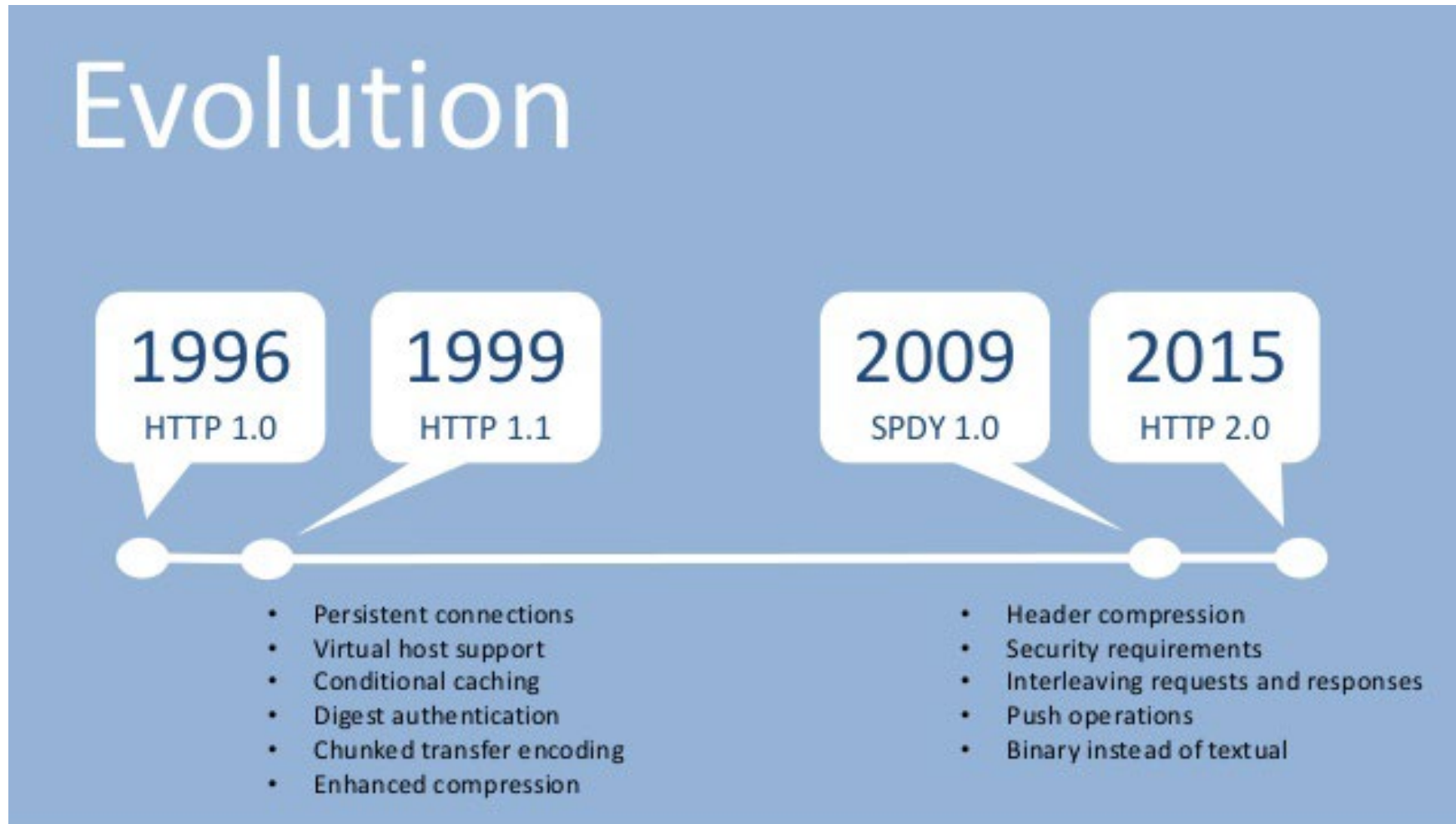
# Introducción



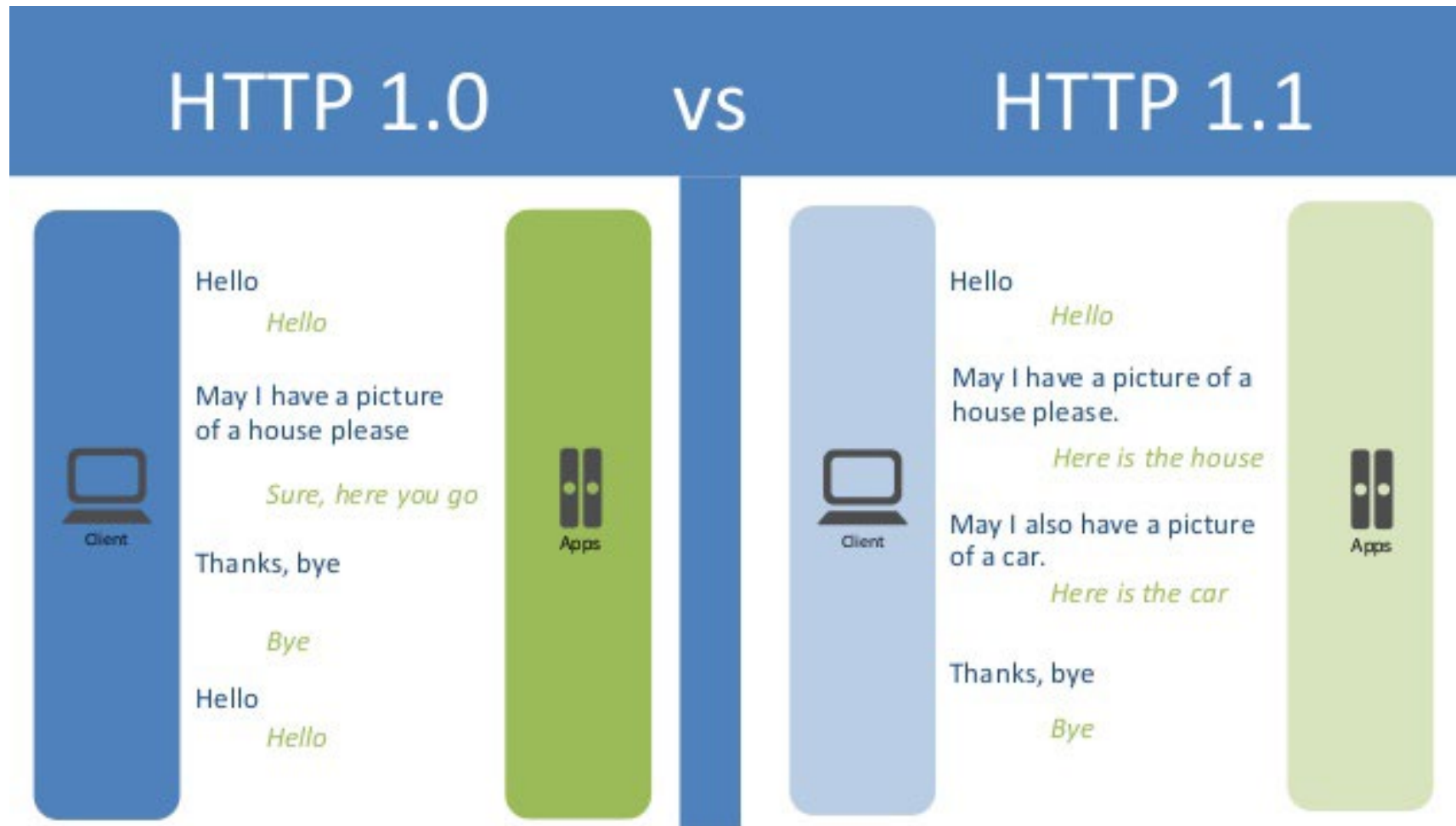
# Evolución

- 1991: HTTP V0.9
- 1996: HTTP V1.0 ([RFC 1945](#))
- 1997: HTTP/1.1 ([RFC 2068](#))
- 1999: Mejoras de HTTP/1.1 ([RFC 2616](#))
- 2007: [HTTP Working Group](#)
- 2009: SPDY protocolo desarrollado por Google para reducir la latencia y que terminaría siendo HTTP/2
- 2014: Mejoras de HTTP/1.1 (RFCs 7230-7235)
- 2015: HTTP/2 ([RFC 7540](#))
- 2018: HTTP/3

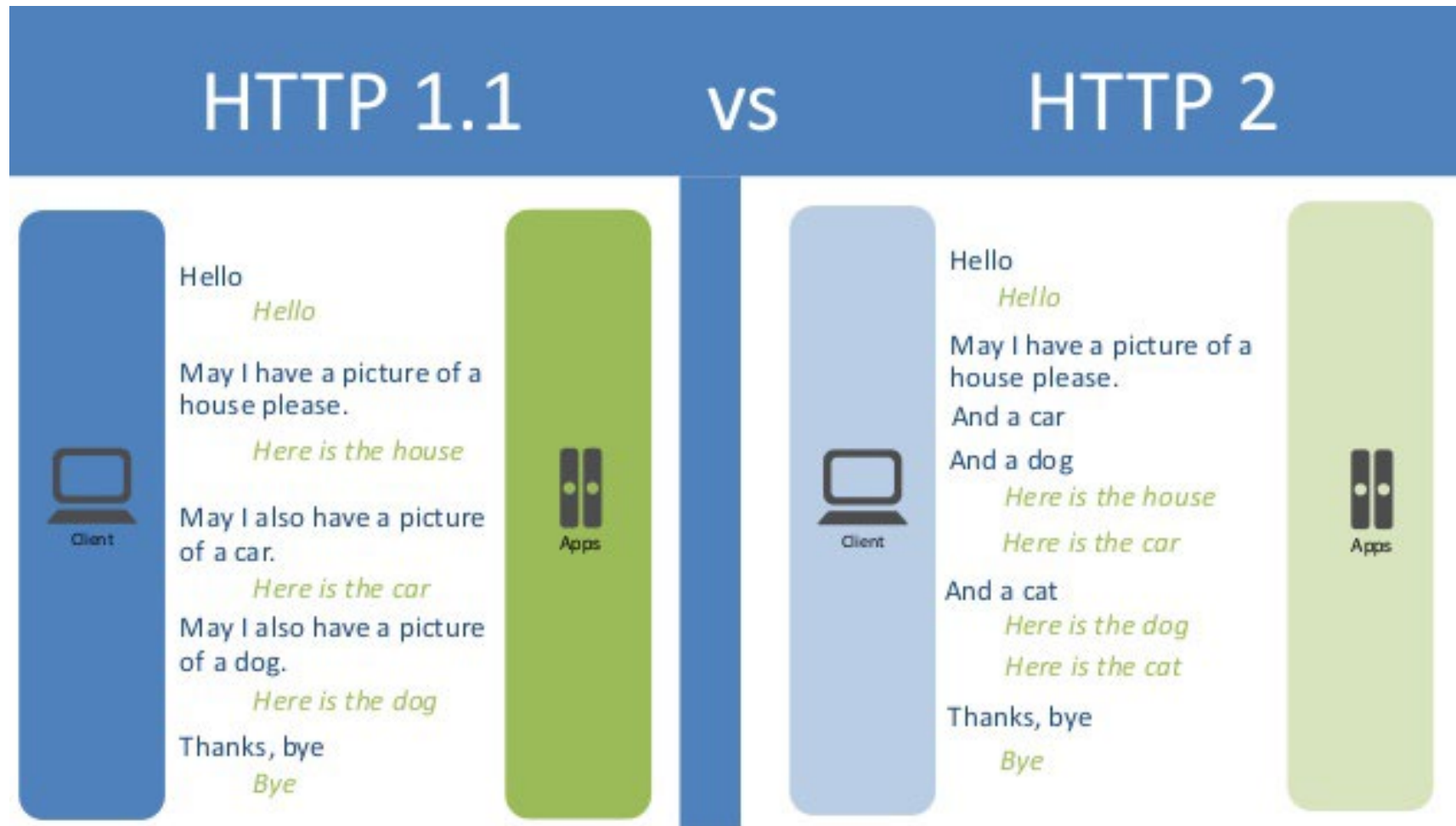
# Evolución



# Evolución



# Evolución



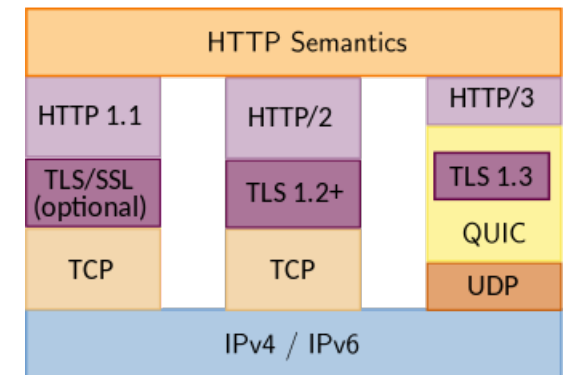


# Evolución

- Uso actual de las versiones:
  - HTTP/2 – 35.1% (Septiembre 2024)
  - HTTP/3 – 31.3% (Septiembre 2024)

fuelle: <https://w3techs.com/>
- Más información de la historia de HTTP

# HTTP/3



- Estándar IETF ([RFC 9114](#))
- Misma semántica que versiones anteriores pero codificado
- Hasta 4 veces más rápido que HTTP/1.1
- Usa QUIC
  - Inicialmente conocido como Quick UDP Internet Connections
  - Capa de transporte
  - Usa UDP en vez de TCP
  - A veces conocido como TCP/2
  - Soluciona un problema de bloqueo de TCP ([head of line blocking](#))
- Soportado por defecto en la mayoría de los navegadores

# CONCEPTOS

# User-agent

- Agente de usuario
- Programa que representa una persona, el cliente (pej. el navegador)




<https://www.whatismybrowser.com/detect/what-is-my-user-agent>

**Your User Agent is:**

Mozilla/5.0 (Windows NT 10.0; Win64; x64)  
AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/86.0.4240.111 Safari/537.36

# User-agent

<http://www.useragentstring.com>

 <b>Chrome 86.0.4240.111</b>	
<b>Mozilla</b>	MozillaProductSlice. Claims to be a Mozilla based user agent, which is only true for Gecko browsers like Firefox and Netscape. For all other user agents it means 'Mozilla-compatible'. In modern browsers, this is only used for historical reasons. It has no real meaning anymore
<b>5.0</b>	Mozilla version
<b>Windows NT 10.0</b>	Operating System:  <a href="#">Windows 10</a>
<b>Win64</b>	(Win32 for 64-Bit-Windows) API implemented on 64-bit platforms of the Windows architecture - currently AMD64 and IA64
<b>x64</b>	64-bit windows version
<b>AppleWebKit</b>	The Web Kit provides a set of core classes to display web content in windows
<b>537.36</b>	Web Kit build
<b>KHTML</b>	Open Source HTML layout engine developed by the KDE project
<b>like Gecko</b>	like Gecko...
<b>Chrome</b>	Name :  <a href="#">Chrome</a>
<b>86.0.4240.111</b>	<a href="#">Chrome</a> version
<b>Safari</b>	Based on Safari
<b>537.36</b>	Safari build
<b>Description:</b>	Free open-source web browser developed by <a href="#">Google</a> . Chromium is the name of the open source project behind <a href="#">Google Chrome</a> , released under the BSD license.
<a href="#">All Chrome user agent strings</a>	

# User-agent

<http://webaim.org/blog/user-agent-string-history/>



NCSA\_Mosaic/2.0  
(Windows 3.1)



NETSCAPE

Mozilla/1.0 (Win3.1)



Mozilla/1.22 (compatible;  
MSIE 2.0; Windows 95)



Mozilla/5.0 (Windows; U;  
Windows NT 5.1; sv-SE;  
rv:1.7.5)  
Gecko/20041108  
Firefox/1.0



Mozilla/5.0 (compatible;  
Konqueror/3.2;  
FreeBSD) (KHTML, like  
Gecko)



Mozilla/5.0 (Macintosh; U;  
PPC Mac OS X; de-de)  
AppleWebKit/85.7  
(KHTML, like Gecko)  
Safari/85.5

# User-agent

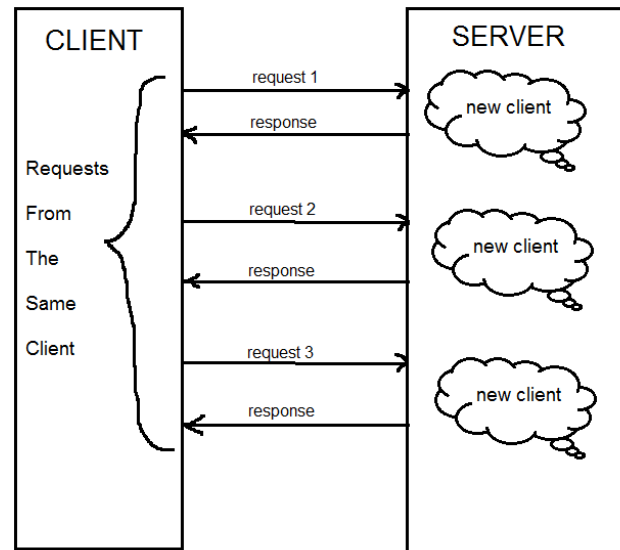
<http://webaim.org/blog/user-agent-string-history/>



Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/525.13 (KHTML, like Gecko)  
Chrome/0.2.149.27 Safari/525.13

# Stateless

- HTTP es un protocolo sin estado
- No se guarda información sobre conexiones anteriores
- La respuesta del servidor es la misma para un cliente previo que para uno nuevo





# ¿Ventajas y desventajas?



# Stateless

- Ventajas
  - Escalabilidad
  - Menos complejidad
  - Mayor rendimiento
  - Se pueden cachear los recursos
- Desventajas
  - Complica la interacción con el usuario
  - Hace falta información extra para mantener la sesión
  - Susceptible a ataques como DDoS (Distributed Denial of Service)

# URL

- Uniform Resource Locator
- Localizador de recursos uniforme
- Recurso (resource)
  - Información que solicita el cliente
  - Pej. documento HTML, imagen, vídeo...
- Referencia a un recurso web especificando la localización en una red y un mecanismo para obtenerlo
- Son únicas, cada URL identifica unívocamente un recurso
- [Living Standard](#)

# URL – Formato

scheme://[userinfo@]host[:port]/path[?query][#fragment]

- **userinfo:** user:password

- **scheme:**

http (HyperText Transport Protocol)

https (HyperText Transport Protocol Secure)

ftp (File Transfer Protocol)

[Y más](#)

- **host:** nombre o la IP del servidor al cual nos queremos conectar

# URL – Formato

`scheme://[userinfo@]host[:port]/path[?query][#fragment]`

- **port:** puerto al cual nos queremos conectar (opcional). Por defecto:  
http → 80  
https → 443  
ftp → 21
- **authority:** `[userinfo@]host[:port]`
- **path:** ruta en el sistema de archivos de la máquina remota donde se encuentra el recurso. La ruta es relativa al directorio raíz de la web.
- **query:** Parámetros extra, normalmente con el formato "q=text"
- **fragment:** Parámetros extra

# URL – Ejemplo

`http://google.es:8080/ejemplo/index.html?q=text#something`

- schema: http
- host: google.es
- port: 8080
- authority: google.es:8080
- path: ejemplo/index.html
- query: q=text
- fragment: something

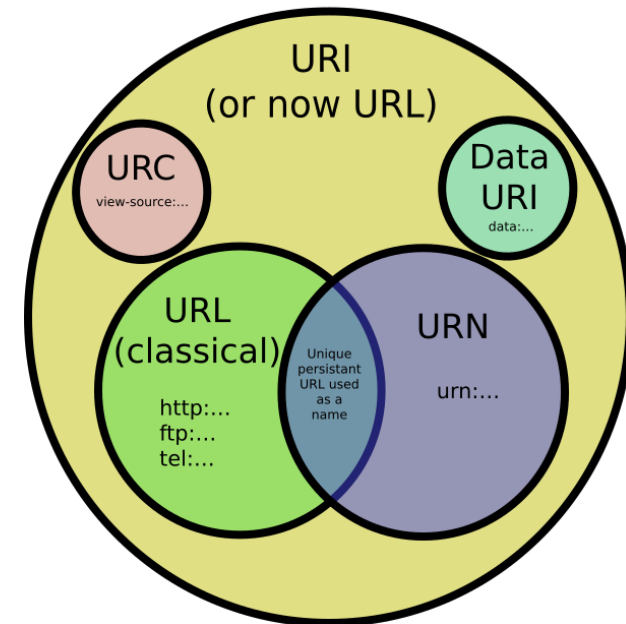
# URL – Ejemplo

Input	<u>Scheme</u>	<u>Host</u>	<u>Port</u>	<u>Path</u>	<u>Query</u>	<u>Fragment</u>
https://example.com/	"https"	"example.com"	null	« the empty string »	null	null
https://localhost:8000/search?q=text#hello	"https"	"localhost"	8000	« "search" »	"q=text"	"hello"
urn:isbn:9780307476463	"urn"	null	null	"isbn:9780307476463"	null	null
file:///ada/Analytical%20Engine/README.md	"file"	null	null	« "ada", "Analytical%20Engine", "README.md" »	null	null

# URL

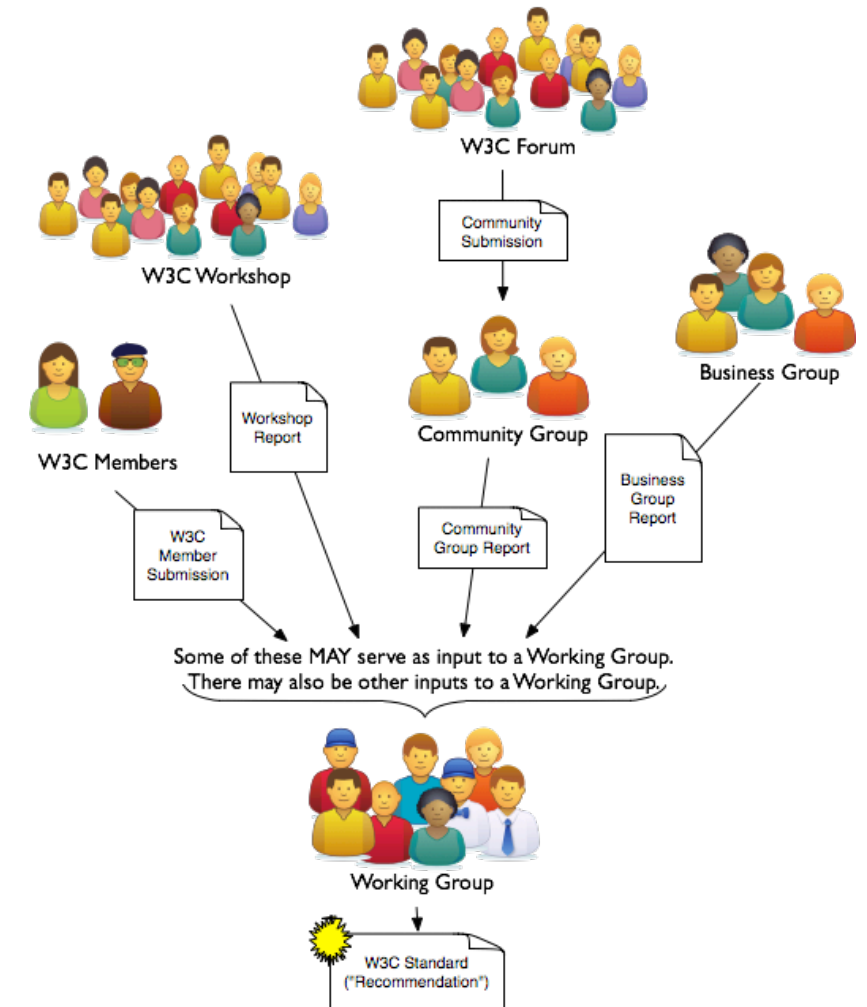
- URI: Uniform Resource Identifier
  - [RFC](#)
  - [Schemes](#)
- URL: Uniform Resource Locator
- URN: Uniform Resource Name
  - urn:isbn:0451450523
  - urn:ISSN:0167-6423
- URC: Uniform Resource Characteristic
  - Metadatos
- [Diferencia entre URI y URL](#)

Venn diagram of URIs as defined by the W3C





# URL



# PETICIÓN HTTP

# Petición HTTP

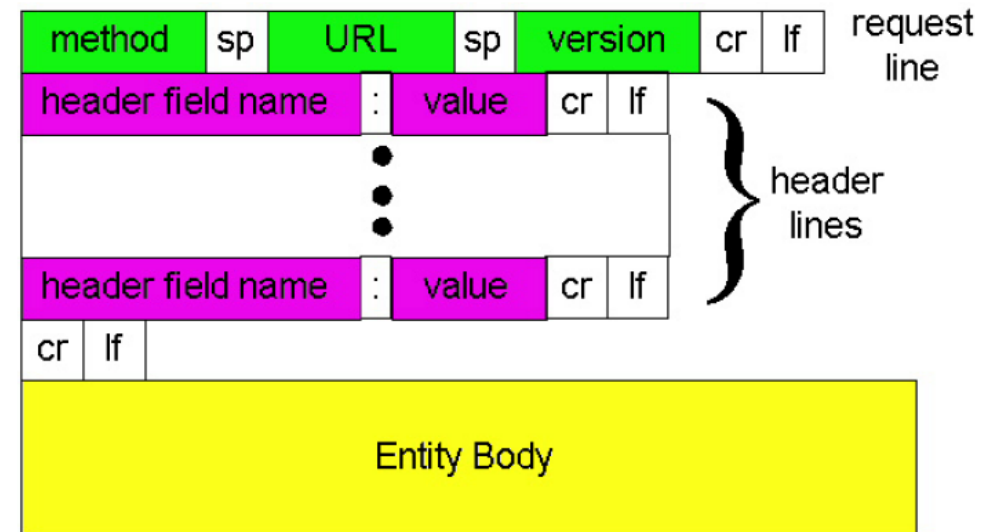
Método SP URL SP Versión Http CRLF

(nombre-cabecera:valor-cabecera(,valor-cabecera)\*CRLF)\*

CRLF

Cuerpo del mensaje

- SP: espacio en blanco
- CRLF: retorno de carro
- (): opcional
- \*: se puede repetir
- <https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html>



# Petición HTTP

- **Método** que se emplea, habitualmente GET o POST
- **URL** del recurso que se está pidiendo
- **Versión** del protocolo HTTP que se está empleando
- Una o más **cabeceras**
  - nombre:valor(,valor)\*
  - Los espacios en blanco forman parte del valor, no son separadores
  - Cada cabecera en una línea

# Petición HTTP

- El **mensaje** puede ser vacío (es el caso del método GET) o no (es el caso del método POST).

```
GET /en/html/dummy?name=MyName&married=not+single&male=yes HTTP/1.1
Host: www.explainth.at
User-Agent: Mozilla/5.0 (Windows;en-GB; rv:1.8.0.11) Gecko/20070312
Firefox/1.5.0.11
Accept: text/xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.explainth.at/en/misc/httpreq.shtml
```

# Petición HTTP

- Esta petición tipo POST sería completamente equivalente a la anterior petición GET en lo que a envío de datos de un formulario al servidor se refiere. En este caso, el contenido del mensaje es sólo la última línea.

```
POST /en/html/dummy HTTP/1.1
Host: www.explainth.at
User-Agent: Mozilla/5.0 (Windows;en-GB; rv:1.8.0.11) Gecko/20070312 Firefox/1.5.0.11
Accept:text/xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5 Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.explainth.at/en/misc/httpreq.shtml
Content-Type: application/x-www-form-urlencoded
Content-Length: 39

name=MyName&married=not+single&male=yes
```

# Petición HTTP

- En el formulario había tres campos, uno con nombre "name", otro con nombre "married", y el último con nombre "male". En la respuesta, cada uno de los tres campos está separado por el símbolo "&". Cada campo va seguido del valor que el usuario ha introducido en el formulario para dicho campo, y separando el nombre del campo y su valor va el signo "=".

Method = GET	Method = POST
<pre>&lt;form method="get"&gt; ... &lt;/form&gt;  GET /suma.html?op1=2&amp;op2=2 HTTP/1.1 . . . Connection: Keep-Alive [blank line]</pre>	<pre>&lt;form method="post"&gt; ... &lt;/form&gt;  POST /suma.html HTTP/1.1 . . Content-Type: ... Content-Length: 11 [blank line] op1=2&amp;op2=2</pre>
El navegador envía los datos ingresados como una cadena de consulta	El navegador envía los datos ingresados en el cuerpo de la solicitud HTTP

# Petición HTTP – Métodos

- También llamados verbos
- Acción que desea efectuar sobre el recurso indicado en la petición
  - Ese recurso podría ser un archivo que reside en un servidor, o podría ser un programa que se está ejecutando en dicho servidor.
- **GET**
  - Obtener un recurso
  - No modifica nada en el servidor
  - Método obligatorio



# Petición HTTP – Métodos

- **HEAD**

- Obtener la cabecera de una petición GET sin el contenido
- Obtención de la meta-información del recurso
- Para conocer el tamaño / versión ...
- Método obligatorio

- **POST**

- Envía datos al servidor para que sean procesados por el recurso especificado en la petición
- Los datos se incluyen en el cuerpo de la petición
- Podría crear un nuevo recurso en el servidor, o actualizar un recurso ya existente

# Petición HTTP – Métodos

- **PUT**

- Envía un recurso determinado (un archivo) al servidor
- A diferencia de POST, este método crea una nueva conexión (socket) y la emplea para enviar el recurso, lo cual resulta más eficiente que enviarlo dentro del cuerpo del mensaje

- **DELETE**

- Elimina el recurso especificado.

- **TRACE**

- Pide al servidor que le envíe un mensaje de respuesta
- Se suele emplear para diagnosticar posibles problemas en la conexión

# Petición HTTP – Métodos

- **OPTIONS**

- Pide al servidor que le indique los métodos HTTP que soporta para una determinada URL

- **CONNECT**

- Se emplea para transformar una conexión ya existente a una conexión encriptada (https).

- **PATCH**

- Modificar parcialmente un recurso ya existente en el servidor

# Petición HTTP – Métodos

- **Safe methods:**
  - Aquellos que no deberían cambiar el estado del servidor. Solo recuperan información
  - Aunque el método GET en principio no debería cambiar nada en la práctica puede utilizarse para enviar comandos al servidor, aunque no está recomendado
- **Idempotent methods:**
  - Múltiples peticiones deberían tener el mismo resultado
  - Los “safe methods” deberían ser por definición también “idempotent methods”
  - De nuevo una mala implementación podría saltarse estas prácticas

# Petición HTTP – Métodos

HTTP method	RFC	Request has Body	Response has Body	Safe	Idempotent	Cacheable
GET	<a href="#">RFC 7231</a>	Optional	Yes	Yes	Yes	Yes
HEAD	<a href="#">RFC 7231</a>	Optional	No	Yes	Yes	Yes
POST	<a href="#">RFC 7231</a>	Yes	Yes	No	No	Yes
PUT	<a href="#">RFC 7231</a>	Yes	Yes	No	Yes	No
DELETE	<a href="#">RFC 7231</a>	Optional	Yes	No	Yes	No
CONNECT	<a href="#">RFC 7231</a>	Optional	Yes	No	No	No
OPTIONS	<a href="#">RFC 7231</a>	Optional	Yes	Yes	Yes	No
TRACE	<a href="#">RFC 7231</a>	No	Yes	Yes	Yes	No
PATCH	<a href="#">RFC 5789</a>	Yes	Yes	No	No	No

[source](#)



# Petición HTTP – Métodos

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data is about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL



# Petición HTTP – Métodos

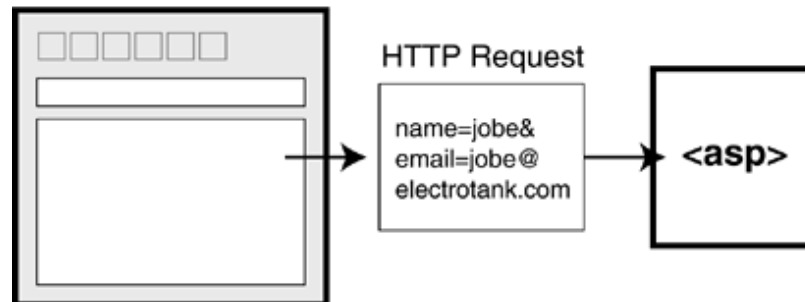
## Using GET

`http://www.somedomain.com/register.asp?name=jobe&email=jobe@electrotank.com`



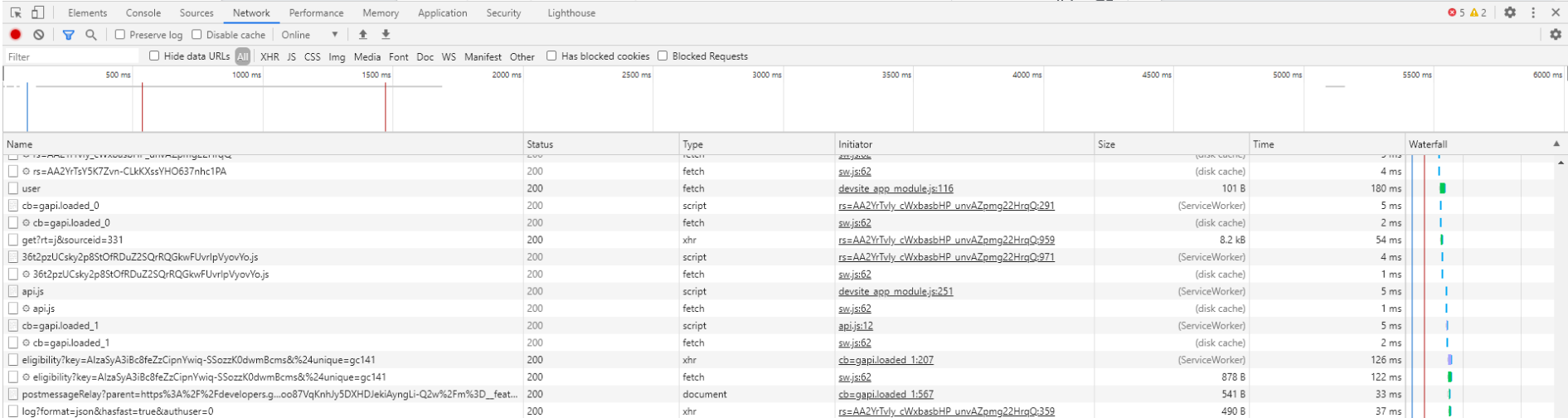
## Using POST

`http://www.somedomain.com/register.asp`



# Chrome DevTools

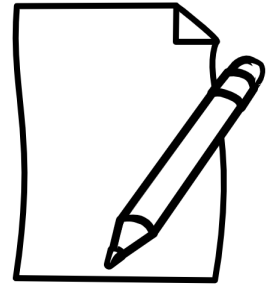
- <https://developer.chrome.com/devtools>
- Ctrl + Shift + J > Network



Name	Status	Type	Initiator	Size	Time	Waterfall
rs=AA2YrTsY5K7Zvm-CLkKXssYHO637nhc1PA	200	fetch	sw.js:62	(disk cache)	4 ms	
user	200	fetch	devsite_app_module.js:116	101 B	180 ms	
cb=gapi.loaded_0	200	script	rs=AA2YrTvlv_cWxbasbHP_unvAZomg22HrgQ:291	(ServiceWorker)	5 ms	
cb=gapi.loaded_0	200	fetch	sw.js:62	(disk cache)	2 ms	
get?rt=j&sourceid=331	200	xhr	rs=AA2YrTvlv_cWxbasbHP_unvAZomg22HrgQ:359	8.2 kB	54 ms	
36t2pzUCsky2p8StOfRDuZ25QrRQGkwFUvripVvovYo.js	200	script	rs=AA2YrTvlv_cWxbasbHP_unvAZomg22HrgQ:371	(ServiceWorker)	4 ms	
36t2pzUCsky2p8StOfRDuZ25QrRQGkwFUvripVvovYo.js	200	fetch	sw.js:62	(disk cache)	1 ms	
api.js	200	script	devsite_app_module.js:251	(ServiceWorker)	5 ms	
api.js	200	fetch	sw.js:62	(disk cache)	1 ms	
cb=gapi.loaded_1	200	script	api.js:12	(ServiceWorker)	5 ms	
cb=gapi.loaded_1	200	fetch	sw.js:62	(disk cache)	2 ms	
eligibility?key=Alza5yA3iBc8feZzCipnYwiq-SSozzK0dwm8cms&%24unique=gc141	200	xhr	cb=gapi.loaded_1:207	(ServiceWorker)	126 ms	
eligibility?key=Alza5yA3iBc8feZzCipnYwiq-SSozzK0dwm8cms&%24unique=gc141	200	fetch	sw.js:62	878 B	122 ms	
postmessageRelay?parent=https%3A%2F%2Fdevelopers.g...oo87VqKnHjy5DXHDJekiAynGLi-Q2w%2Fm%3D_feat...	200	document	cb=gapi.loaded_1:567	541 B	33 ms	
log?format=json&hasfast=true&authuser=0	200	xhr	rs=AA2YrTvlv_cWxbasbHP_unvAZomg22HrgQ:359	490 B	37 ms	



# HTTP – Ejercicio 1



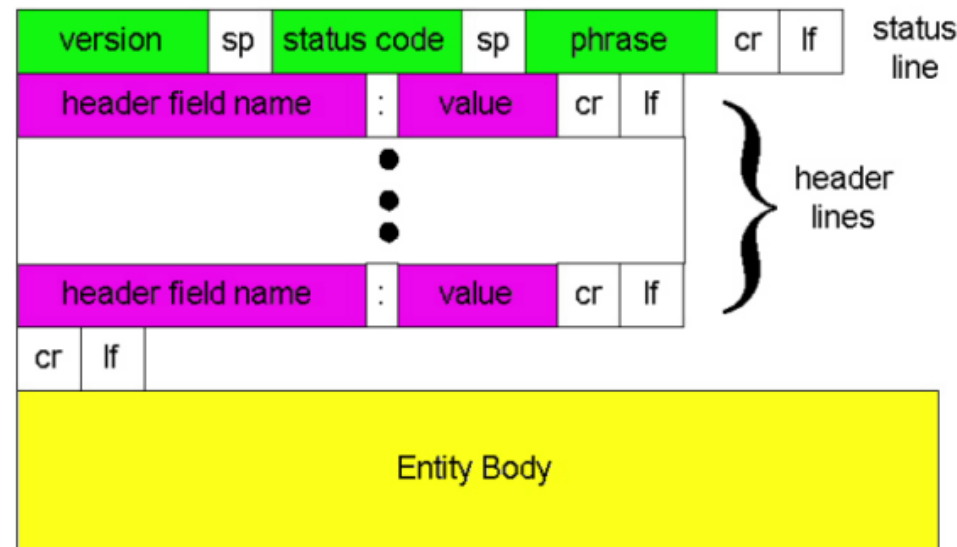
- Busca una web que tenga un formulario
  - Abre navegación privada
  - Busca alguna página para logearse / registrarse o un formulario de contacto
- Analiza los mensajes HTTP que hay
  - ¿Identificas alguno?
- Rellena el formulario con cualquier dato y dale a submit
- Analiza el mensaje HTTP que se ha generado con las herramientas del navegador
- Repite los pasos editando el HTML y cambiando el método

# RESPUESTA HTTP

# Respuesta HTTP

Una respuesta del servidor en el protocolo http sigue la siguiente estructura:

Versión-http SP código-estado SP frase-explicación CRLF  
(nombre-cabecera: valor-cabecera ("," valor-cabecera)\* CRLF)\*  
CRLF  
Cuerpo del mensaje



# Respuesta HTTP

- **Código de estado:** indica si la petición ha tenido éxito o habido algún error con ella
- **Frase:** explicación del código
- **Cabeceras:** misma estructura que en las peticiones
- **Cuerpo:** la respuesta del servidor

# Respuesta HTTP

HTTP/1.1 200 OK

Content-Type: text/xml; charset=utf-8

Content-Length: length

<html>

<head> <title> Título de nuestra primera página </title> </head>

<body>

¡Hola mundo!

</body>

</html>

# Respuesta HTTP – Códigos de estado

- Números de tres dígitos
- Forman parte de las respuestas HTTP
- Explican qué ha sucedido al intentar llevar a cabo una petición
- Varias categorías:
  - **1xx**: Mensajes
  - **2xx**: Operación realizada con éxito
  - **3xx**: Redirección
  - **4xx**: Error por parte del cliente
  - **5xx**: Error del servidor
- [Listado](#)

# Respuesta HTTP – Códigos de estado

## Códigos 1xx: Información

- 100 Continue: Conexión aceptada, continua la petición
- 101 Switching Protocols: Cambiando protocolos
- 102 Processing: Procesando todavía la petición

# Respuesta HTTP – Códigos de estado

**Códigos 2xx:** Operación realizada con éxito

- 200 OK: La petición se ha realizado con éxito
- 201 Created: Petición OK y se ha creado un nuevo recurso
- 202 Accepted: Petición aceptada pero el proceso no ha terminado todavía
- 204 No Content: Sin Contenido
- 205 Reset Content: Le indica al user agent que recargue el documento
- 206 Partial Content: Contenido parcial





# Respuesta HTTP – Códigos de estado

## Códigos 3xx: Redirección

- 301 Moved Permanently: contiene la nueva URL
- 302 Found: El cambio es temporal
- 304 Not modified: No modificado, se puede usar la versión cacheada
- 307 Temporary Redirect: Muy parecido al 302
- 308 Permanent Redirect: Muy parecido al 301

# Respuesta HTTP – Códigos de estado

## Códigos 4xx: Error por parte del cliente

- 400 Bad Request: Petición malformada o inválida
- 401 Unauthorized: El cliente tiene que registrarse
- 403 Forbidden: El cliente no tiene los derechos de acceso
- 404 Not Found: No existe el recurso indicado
- 405 Method Not Allowed: Ese método HTTP no está permitido
- 409 Conflict: Por ejemplo, múltiples ediciones simultaneas
- 410 Gone: El recurso ha sido borrado permanentemente
- [418 I'm a teapot](#): broma de April's fool



# Respuesta HTTP – Códigos de estado

## **Códigos 5xx:** Error del servidor

- 500 Internal Server Error: el servidor no sabe gestionar la situación
- 502 Bad Gateway
- 503 Service Unavailable: pej. sobrecargado o en mantenimiento
- 504 Gateway Timeout

## Y muchos más códigos






- Además hay códigos no oficiales

# Respuesta HTTP – Códigos de estado


## AN SEO'S GUIDE TO HTTP STATUS CODES


Every web page you visit returns a status code, to give the browser additional information and instructions. Search bots see these codes, and some of them can impact SEO. Here are a few of the big ones:


### CAST OF CHARACTERS


 The Visitor    The Robot    Link Juice     The Pages


### HTTP STATUS CODES


**200**  **OK/Success** Everyone arrives at Page A. There is much rejoicing!

**301**  **Permanent\*** Everyone is redirected to the new location, Page B.

**302**  **Temporary\*** Visitors and bots are redirected. Juice is left behind.


**404**  **Not Found** Original page is gone. Visitors may see a 404 page.

**500**  **Server Error** No page is returned. Everyone is lost and confused :(

**503**  **Unavailable** Asks everyone to come back later. A 404 alternative.

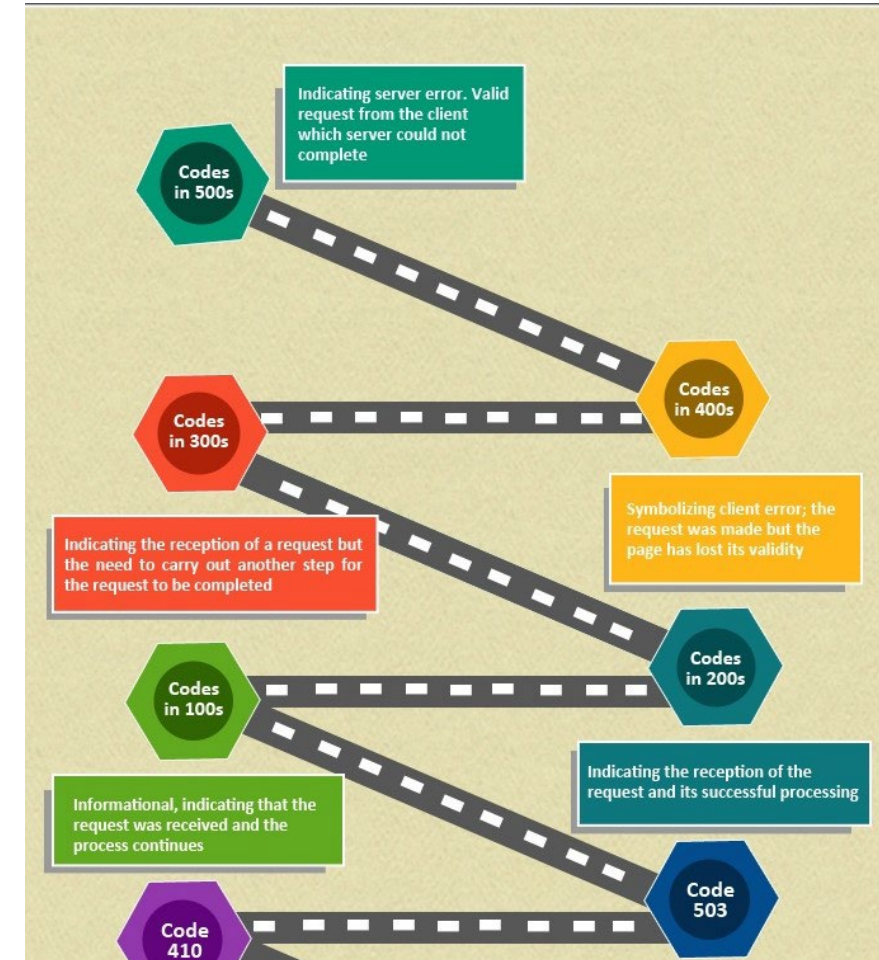
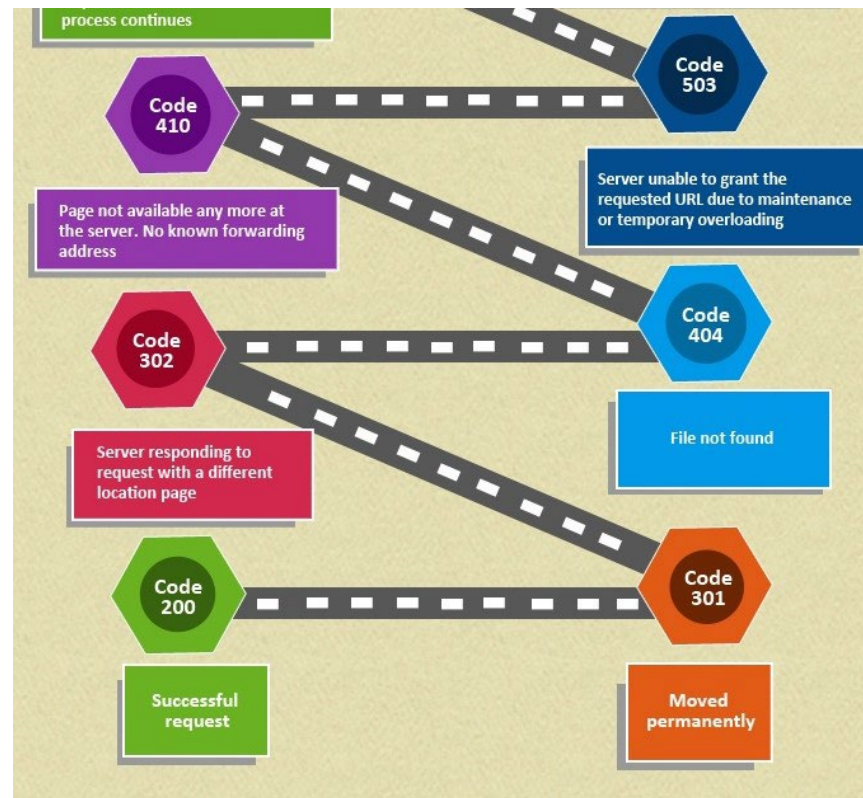
\* Technically, code 301 is "Moved Permanently" and 302 is "Found", but SEOs refer to them as "Permanent Redirect" and "Temporary Redirect".

### THE CANONICAL TAG

**REL**  **Canonical** Alternative to 301-redirects. Visitors still see Page A.

Copyright © 2011 SEOmoz, Inc. ([www.seomoz.org](http://www.seomoz.org)). All Rights Reserved.

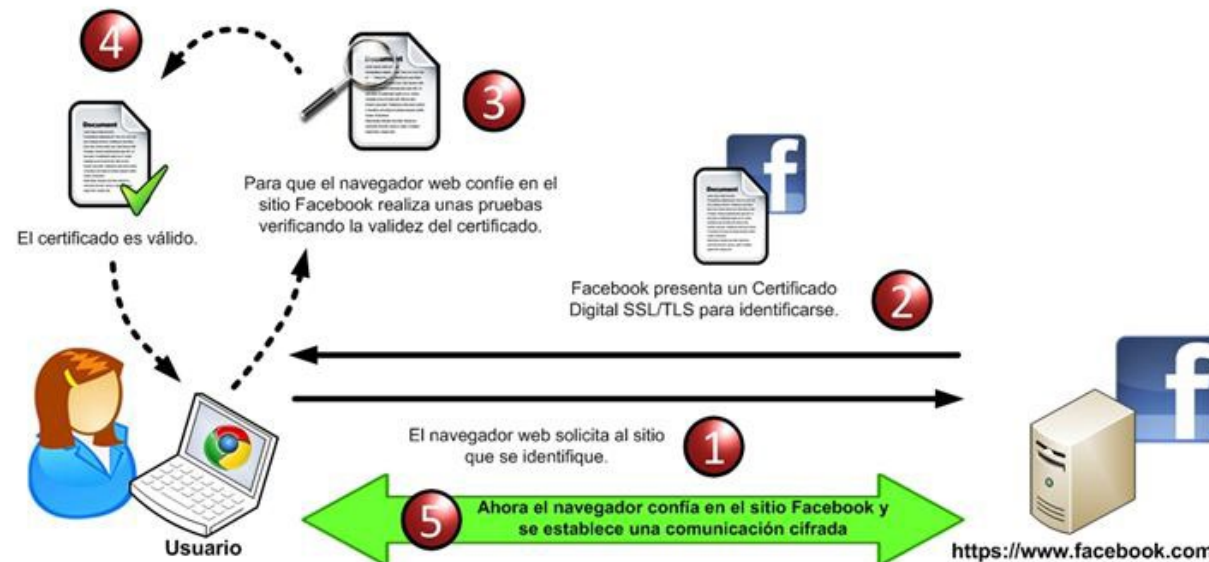
# Respuesta HTTP – Códigos de estado



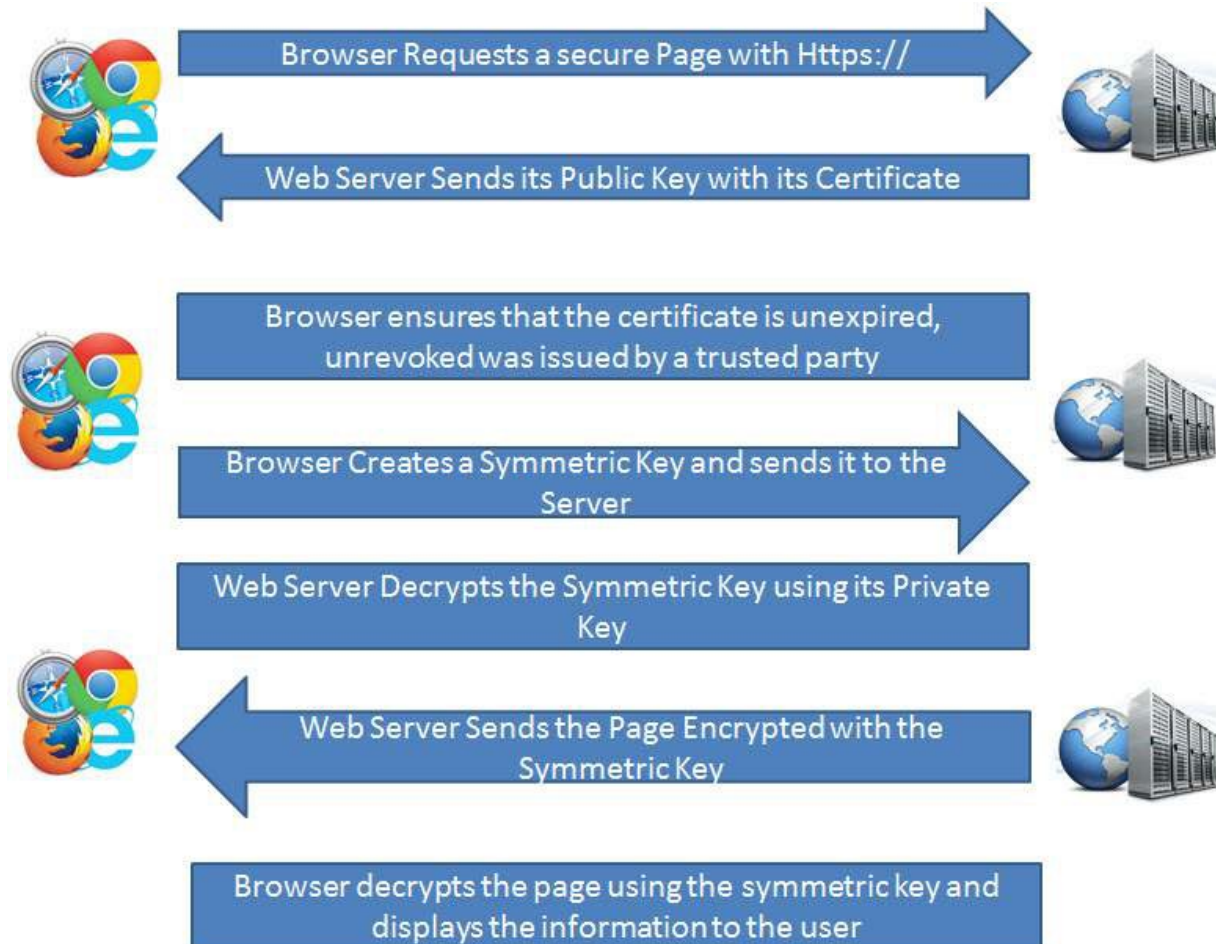


# HTTPS

- Hypertext Transfer Protocol Secure
- Protocolo de aplicación basado en el protocolo HTTP
- Destinado a la transferencia segura de datos.
- Utiliza un cifrado basado en SSL/TLS (puerto 443)



# HTTPS

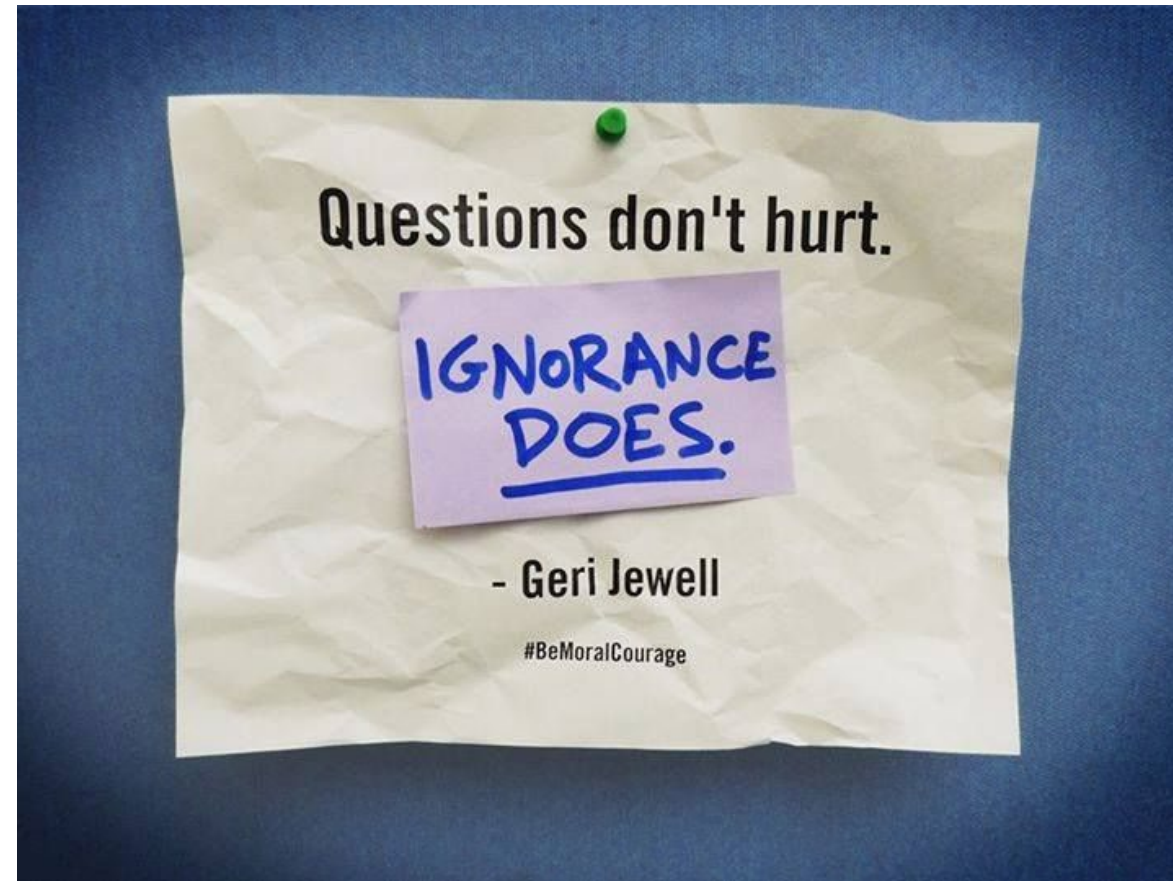


# Postman

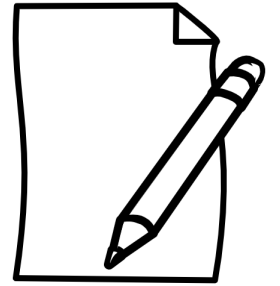
- Plataforma de desarrollo API
  - Enviar peticiones HTTP
  - Testing
  - Simular endpoints
- Antes era una extensión del navegador
- Ahora es una app standalone
- También se puede usar desde el navegador:  
<https://go.postman.co/build>
- Web: <https://www.postman.com/>



# Dudas



# HTTP – Ejercicio 2



- Instala Postman o úsalo desde el navegador
- Prueba a usar cada uno de los 9 métodos HTTP
  - Analiza los requests y responses
- Prueba alguno de los códigos de estado en: <https://httpstat.us/>

# Referencias

- HTTP - Hypertext Transfer Protocol
  - <https://www.w3.org/Protocols/>
  - <https://www.w3.org/Protocols/History.html>
  - <https://www.w3.org/Protocols/Classic.html>
  - <https://developer.mozilla.org/en-US/docs/Web/HTTP>
- Artículos sobre HTTP
  - <https://code-maze.com/http-series/>
- E-Book
  - Introduction to HTTP (<https://launchschool.com/books/http>)
- Gourley, D., Totty, B., Sayer, M., Aggarwal, A., & Reddy, S. (2002). HTTP: the definitive guide. " O'Reilly Media, Inc."