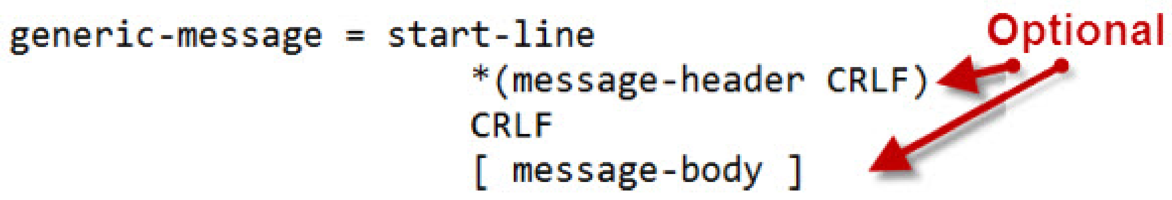
HTTP & API

> Hypertext Transfer Protocol: For transferring data across the web

- All requests originate at the client (your browser)
- The server responds to a request.
- The requests(commands) and responses are in readable text.
- The requests are independent of each other and the server **doesn't need to track** the requests.



```
generic-message = start-line
                  *(message-header CRLF)
                  CRLF
                  [ message-body ]
```

HTTP Request or Response Message Format

HTTP Request

Method + Resource Path + protocol version

Start-line

`GET /test.htm HTTP/1.1`

- **GET** is the method
- **/testpage.htm** is the relative path to the resource.
- **HTTP/1.1** is the protocol version we are using

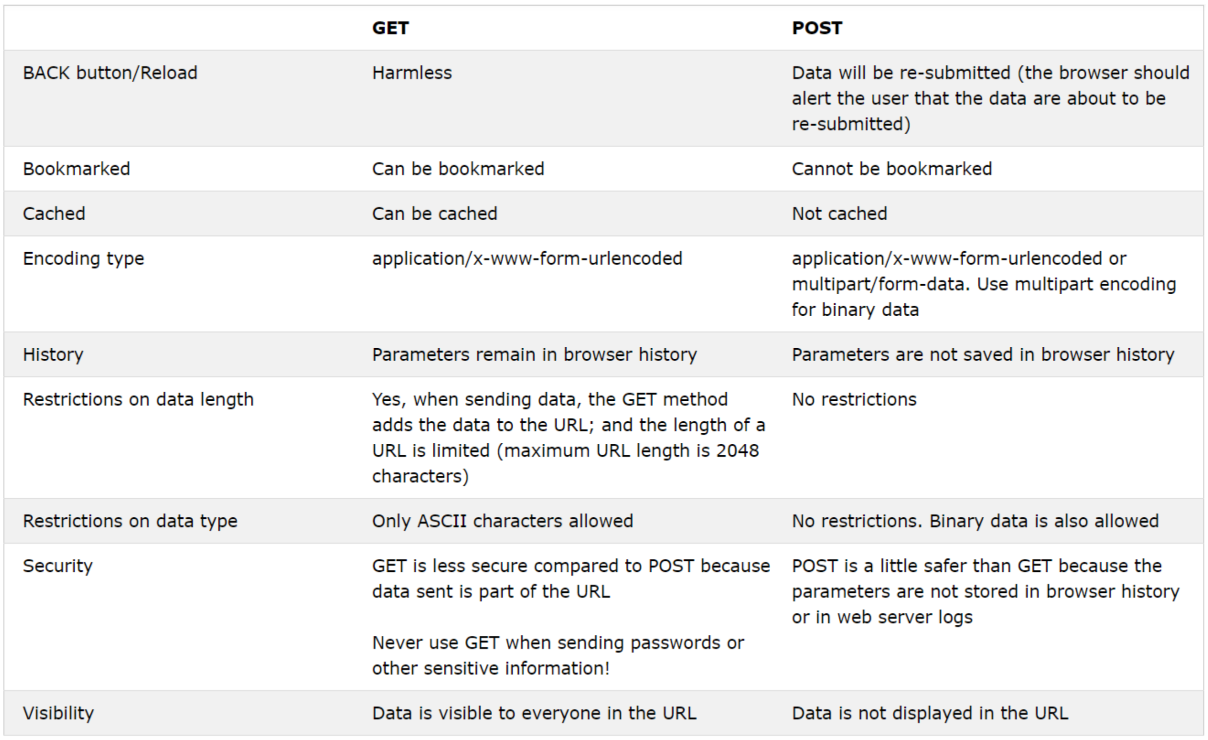
URL (uniform resource locator) we enter creates the relative URI (uniform resource indicator)

The GET Method

GET is used to **request data** from a specified resource. Resources should be fetched

The POST Method

POST is used to **send data** to a server to create/update a resource. Resources should be pushed to the server



	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are 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

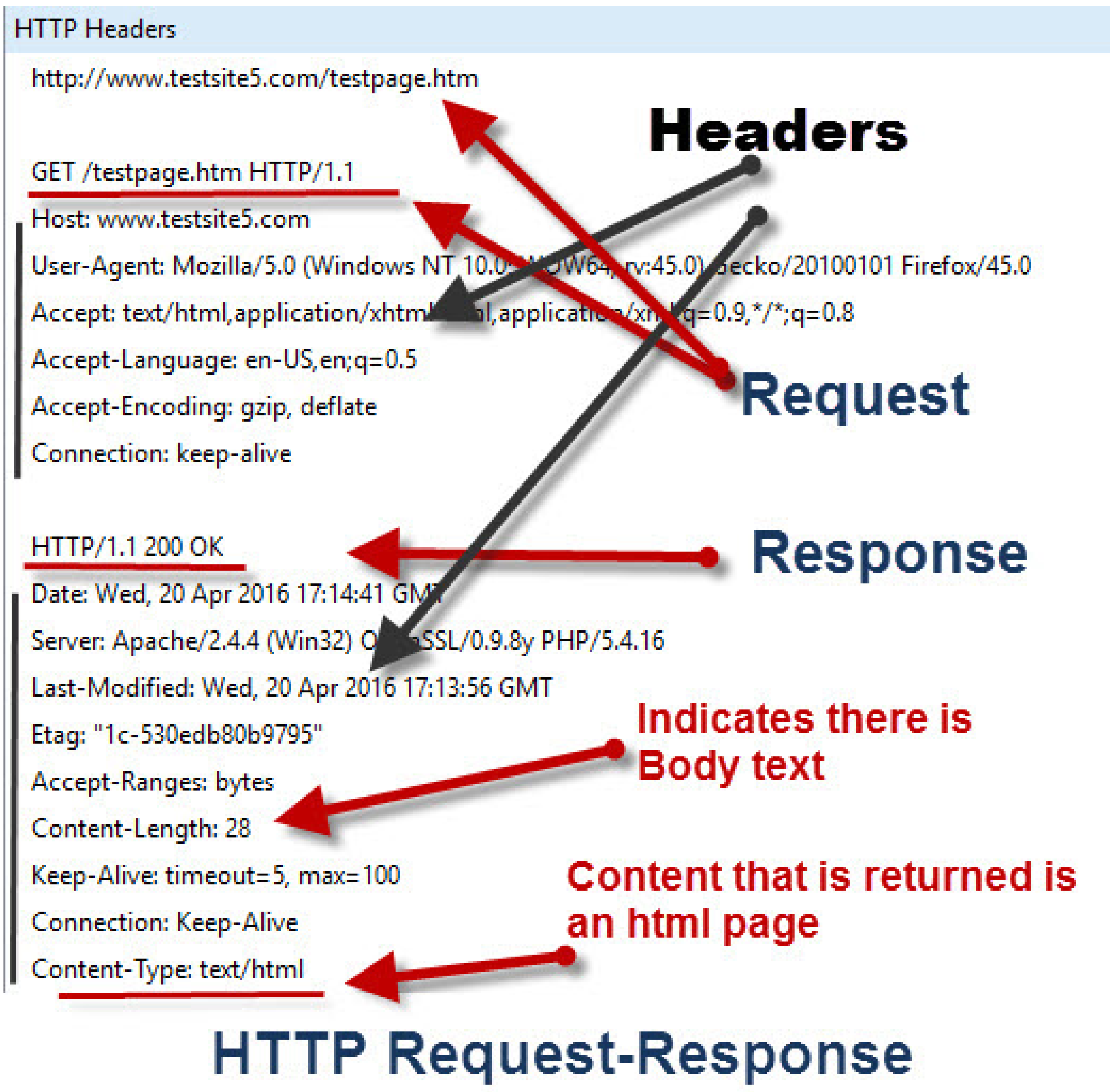
HTTP Response

Each request has a response

- **STATUS code** And Description
- 1 or more optional headers
- Optional Body message can be many lines including binary data

Response Status codes are split into 5 groups each group has a meaning and a three digit code.

- **1xx** – Informational
- **2xx** – Successful
- **3xx** -Multiple Choice
- **4xx**– Client Error
- **5xx** -Server Error



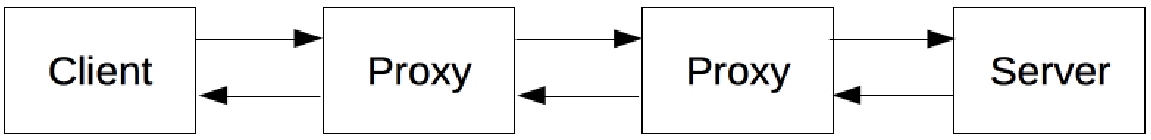
```
HTTP Headers
http://www.testsite5.com/testpage.htm

GET /testpage.htm HTTP/1.1
Host: www.testsite5.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Date: Wed, 20 Apr 2016 17:14:41 GMT
Server: Apache/2.4.4 (Win32) OpenSSL/0.9.8y PHP/5.4.16
Last-Modified: Wed, 20 Apr 2016 17:13:56 GMT
Etag: "1c-530edb80b9795"
Accept-Ranges: bytes
Content-Length: 28
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Headers
Request
Response
Indicates there is Body text
Content that is returned is an html page
HTTP Request-Response

Each individual request is sent to a **server**, which will handle it and provide an answer, called the response. Between this **request** and **response** there are numerous entities, collectively designated as `proxies`, which perform different operations and act as gateways or caches, for example.



The browser is always the entity initiating the request.

To present a Web page, the browser sends an original request to fetch the HTML document from the page. It then parses this file, fetching additional requests corresponding to execution scripts, layout information (CSS) to display, and sub-resources contained within the page (usually images and videos). The Web browser then mixes these resources to present to the user a complete document, the Web page.

Hypertext

This means some parts of displayed text are links which can be activated (usually by a click of the mouse) to fetch a new Web page, allowing the user to direct their user-agent and navigate through the Web.

Proxies

- caching (the cache can be public or private, like the browser cache)
- filtering (like an antivirus scan, parental controls, …)
- load balancing (to allow multiple servers to serve the different requests)
- authentication (to control access to different resources)
- logging (allowing the storage of historical information)

HTTP is stateless, but HTTP cookies allow stateful sessions

HTTP is stateless: there is no link between two requests being successively carried out on the same connection. Using header extensibility, HTTP Cookies are added to the workflow, allowing session creation on each HTTP request to share the same context, or the same state.

TCP connection

Before a client and server can exchange an HTTP request/response pair, they must establish a TCP connection, a process which requires several round-trips.

Features controllable with HTTP

- *Cache* How documents are cached can be controlled by HTTP. The server can instruct proxies, and clients, what to cache and for how long. The client can instruct intermediate cache proxies to ignore the stored document.
- *Relaxing the origin constraint* To prevent snooping and other privacy invasions, Web browsers enforce strict separation between Web sites. Only pages from the **same origin** can access all the information of a Web page. Though such constraint is a burden to the server, HTTP headers can relax this strict separation server-side, allowing a document to become a patchwork of information sourced from different domains (there could even be security-related reasons to do so).
- *Authentication* Some pages may be protected so only specific users can access it. Basic authentication may be provided by HTTP, either using the `WWW-Authenticate` and similar headers, or by setting a specific session using HTTP cookies.
- *Proxy and tunneling* Servers and/or clients are often located on intranets and hide their true IP address to others. HTTP requests then go through proxies to cross this network barrier. Not all proxies are HTTP proxies. The SOCKS protocol, for example, operates at a lower level. Others, like ftp, can be handled by these proxies.
- *Sessions*Using HTTP cookies allows you to link requests with the state of the server. This creates sessions, despite basic HTTP being a state-less protocol. This is useful not only

for e-commerce shopping baskets, but also for any site allowing user configuration of the output.

HTTP Flow

1. open a TCP connection (HTTP/1.1 now no need to reopen connection)
2. send an HTTP message

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

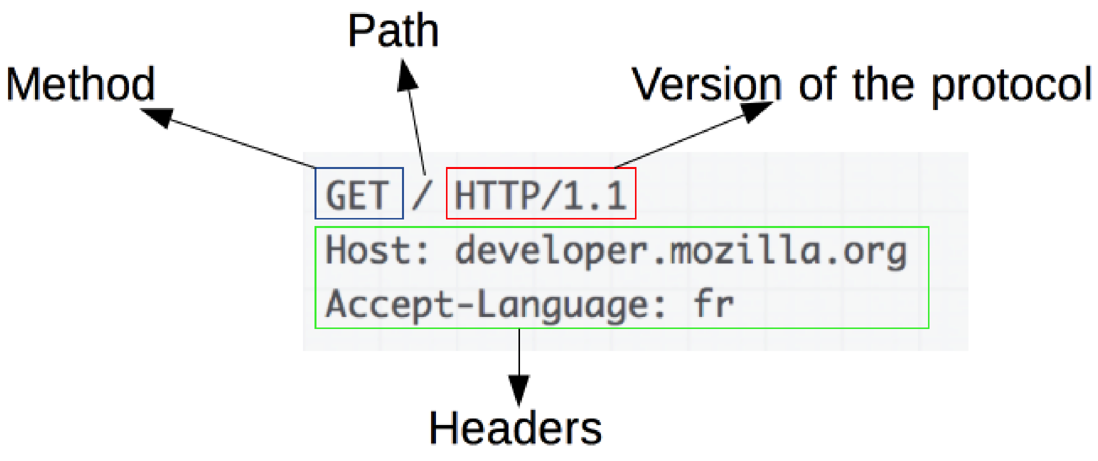
3. read the response sent by the server

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)
```

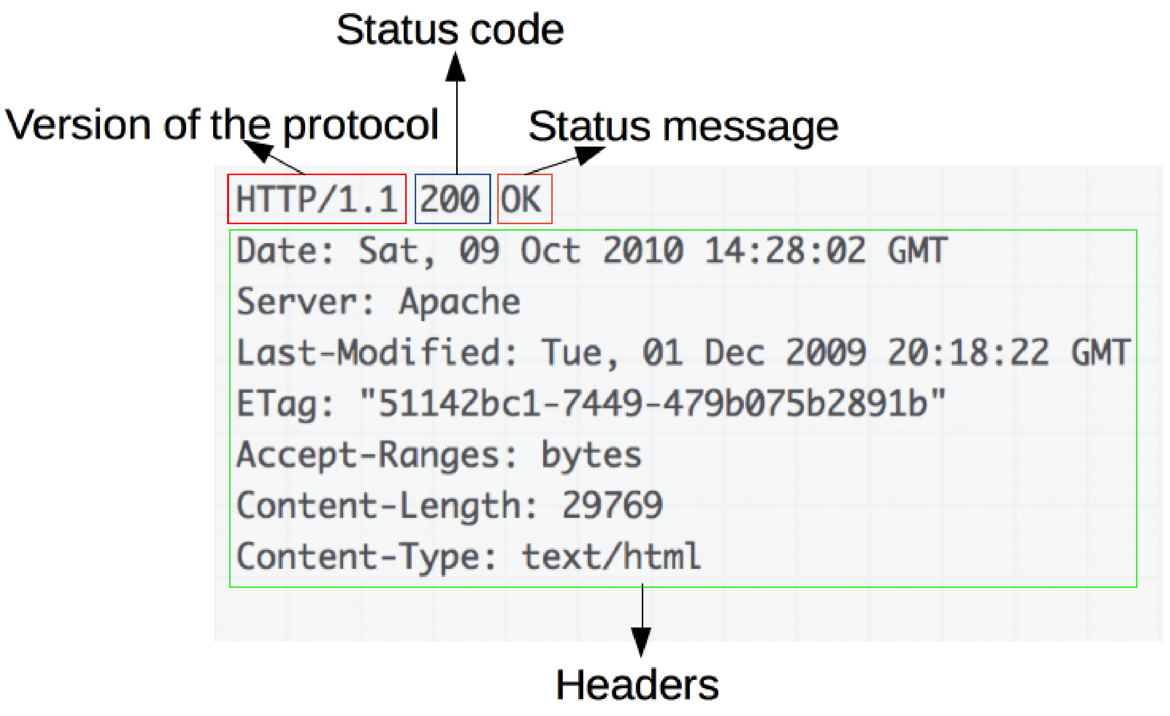
4. close connection and reuse it for further requests

HTTP request



```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

HTTP response



```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

HTTP/1.1: Status Code Definitions
https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Common port for HTTP

`80`: Hypertext Transfer Protocol (**HTTP**) used in the World Wide Web

Endpoint of communication (server side). Ports are identified for each protocol and address combination.

HTTP caches

Caching is a technique that stores a **copy** of a given resource and serves it back when requested. When a web cache has a requested resource in its store, it **intercepts** the request and returns its copy instead of re-downloading from the originating server.

There are several kinds of caches: these can be grouped into two main categories: **private or shared caches.**

common HTTP caches are typically limited to caching responses to `GET` and may decline other methods.

control cache: `Cache-Control`

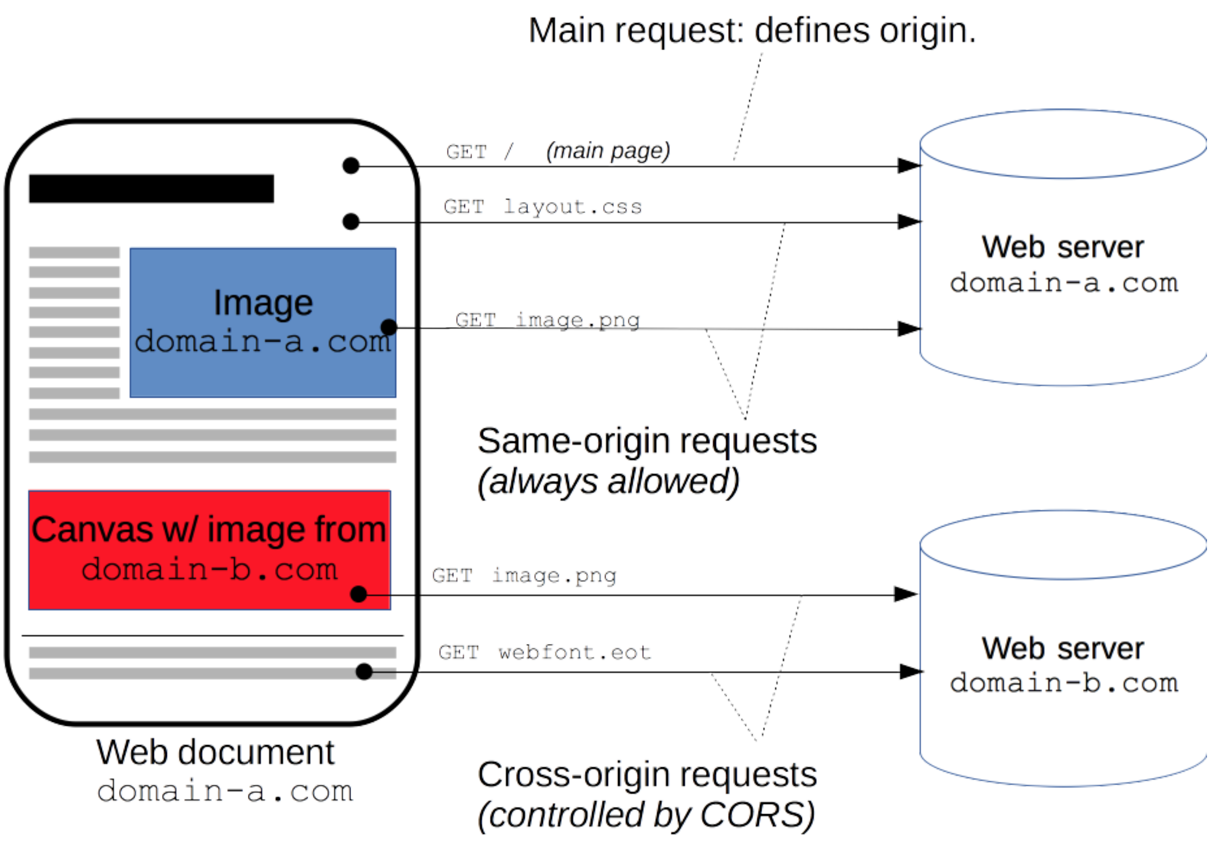
```
Cache-Control: no-store   No caching
Cache-Control: no-cache   a cache send request to origin server for validation before releasing copy
```

```
Cache-Control: private    for a single user only
Cache-Control: public     can be shared by others
Cache-Control: max-age=31536000 expiration in <second>
```

HTTP Cross-Origin Resource Sharing (CORS)

uses additional HTTP headers to let web app run at one origin have permission to access selected resources from a server at a different origins.

An example of a cross-origin request: The frontend JavaScript code for a web application served from `http://domain-a.com` uses `XMLHttpRequest` to make a request for `http://api.domain-b.com/data.json .`


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Web server domain-a.com
Image domain-a.com
Same-origin requests (always allowed)
Canvas w/ image from domain-b.com
GET image.png
GET webfont.eot
Web server domain-b.com
Web document domain-a.com
Cross-origin requests (controlled by CORS)

The World Wide Web

Built over the existing TCP and IP protocols, it consisted of 4 building blocks:

- A textual format to represent hypertext documents, the *HyperText Markup Language*(HTML).
- A simple protocol to exchange these documents, the *HypertText Transfer Protocol* (HTTP).
- A client to display (and accidentally edit) these documents, the first Web browser called *WorldWideWeb*.
- A server to give access to the document, an early version of *httpd*.

HTTP/1.1: standardized protocol

- A **connection** can be **reused**, saving time to reopen to display resources
- **Pipelining** added: allow to send second request before the answer for the first is fully transmitted
- **Chunked responses** supported
- **Additional cache control mechanisms**
- **Content negotiation**, including language, encoding, or type
- allow a client and a server to agree on the most adequate content to exchange
- `Host` **heade**r: ability to host different domains at the same IP address now allows server collocation

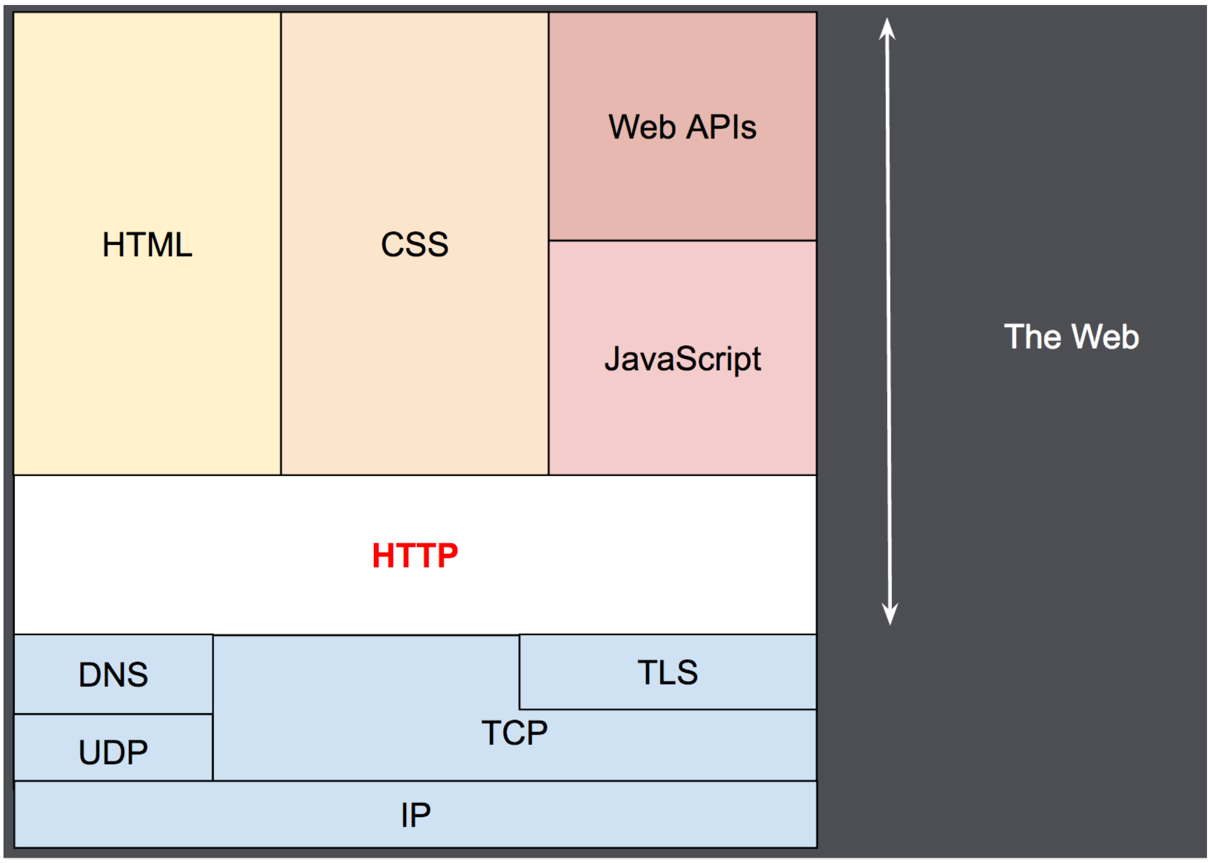
---

API: Application Programming Interface

Messenger takes request and tell system what you want and return the response

- A set of routines, protocols, and tools for building software applications
- defines functionalities that are independent of their respective implementations
- A good API make it easier to develop a program by providing all the building blocks
- often come in the form of a library that includes specifications for routines, data structures, object classes, and variables

---



HTTP flow in details

1. Establishing a connection

- Open a connection in the underlying transport layer, usually this is **TCP**
- default port is 80, other common ports are 8000 or 8080
- URL of the page to be fetched contains both domain name and port number

2. Sending a client request

- connection established, web browser at client side send the request
  1. request method followed by parameters (path of document, HTTP protocol version)
  2. HTTP header, giving server information about what type of data is appropriate
  3. optional data block, may contain further data mainly used by POST method
  - Example:

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

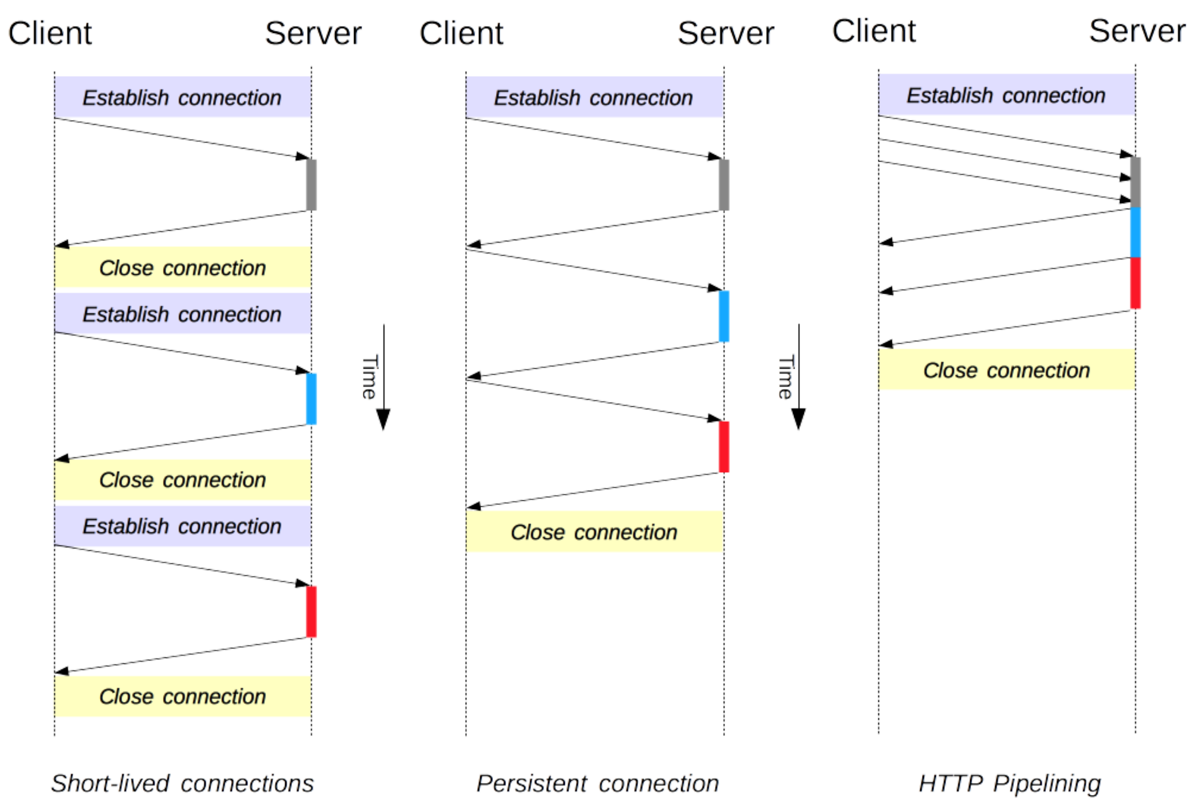
3. Server process request and return response

- server response is formed of:
  1. status line: acknowledgement of HTTP version used, followed by status request (readable text)
     - some status codess
     - `200`: **OK**. The request has succeeded.
     - `301`: **Moved Permanently**. This response code means that the URI of requested resource has been changed.
     - `404`: **Not Found**. The server cannot find the requested resource.
  2. specific HTTP headers, giving client information about data sent (type, data size, compression algorithm used, caching, etc.)
  3. data block, contains optional data
  - Example:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 55743
Connection: keep-alive
Cache-Control: s-maxage=300, public, max-age=0
Content-Language: en-US
Date: Thu, 06 Dec 2018 17:37:18 GMT
ETag: "2e77ad1dc6ab0b53a2996dfd4653c1c3"
Server: meinheld/0.6.1
Strict-Transport-Security: max-age=63072000
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Vary: Accept-Encoding,Cookie
Age: 7


<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>A simple webpage</title>
</head>
<body>
  <h1>Simple HTML5 webpage</h1>
  <p>Hello, world!</p>
</body>
</html>
```

HTTP Connection Managements



- Persistent connection
    - `Keep-Alive` header to specify minimum time connection kept open
- Pipelining
    - Only `GET` , `HEAD` , `PUT` ,and `DELETE` can be replayed safely.

More about request methods

- **Idempotent (PUT, DELETE): do multiple times is fine**
- **POST is not idempotent, do multiple times will create multiple entries**

`GET` : read content, fetch data but not changing data from the server

The `GET` method requests a representation of the specified resource. Requests using `GET` should only **retrieve data**.

`POST` : create content, changing data in the server database

The `POST` method is used to **submit an entity** to the specified resource, often causing a change in **state** or side effects **on the server**.

`PUT` : update content, changing data in the server database

The `PUT` method **replaces all current representations of the target resource** with the request payload.

`DELETE` **: delete content, changing data in the server database**

The `DELETE` method **deletes** the specified resource.

`HEAD`

The `HEAD` method asks for a response identical to that of a `GET` request, but **without the response body**.

`CONNECT`

The `CONNECT` method **establishes a tunnel** to the server identified by the target resource.

`OPTIONS`

The `OPTIONS` method is used to describe the **communication options** for the target resource.

`TRACE`

The `TRACE` method performs a **message loop-back test** along the path to the target resource.

`PATCH`

The `PATCH` method is used to apply **partial modifications to a resource.**