Hypertext Transfer Protocol

> "The Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser."

Source: Hypertext Transfer Protocol - https://en.wikipedia.org

> "Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests."

Source: HTTP - https://developer.mozilla.org/en-US/docs/Web/HTTP

So, HTTP is a protocol which facilitates exchange of data between client and server over the web. This data is usually in the form of hypertext.

The client and server communicate by exchanging individual messages, known as HTTP messages. There are 2 types of messages: *requests* sent by client to trigger an action on a server, and *responses*, the answer from the server.

HTTP requests, and responses, share similar structure and are composed of:

1. A *start-line* describing the requests to be implemented, or its status of whether successful or a failure. This start-line is always a single line.
2. An optional set of *HTTP headers* specifying the request, or describing the body included in the message.
3. A blank line indicating all meta-information for the request has been sent.
4. An optional *body* containing data associated with the request (like content of an HTML form), or the document associated with a response. The presence of the body and its size is specified by the start-line and HTTP headers.

The start-line and HTTP headers of the HTTP message are collectively known as the *head* of the requests, whereas its payload is known as the *body*.

HTTP Requests:

Sent by clients to initiate an action on the server. Their start line contains 3 elements:

1. An HTTP Method (verb like GET, POST or PUT or noun like HEAD or OPTIONS) that describes action to be performed.

2. The *request target* (a URL or the absolute part of protocol, port, and domain)
3. HTTP version

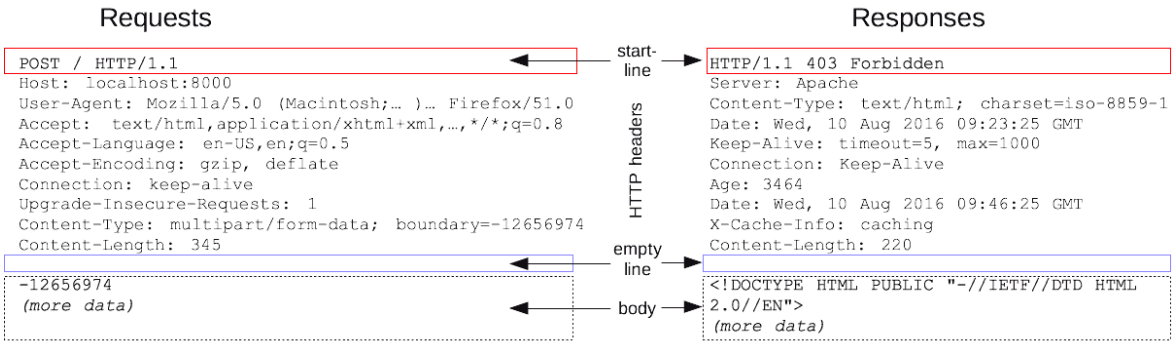
HTTP Headers consist of a case insensitive string followed by a colon (':') and a value. Not all requests have a body. Some requests send data to server to update it.

HTTP Responses:

The start line of HTTP response contains the following:

1. The protocol version.
2. The status code, indicating success or failure.
3. A *status text*.

The header and body part of response is similar to that of request.



Source: HTTP Messages - https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

HTTP Request Methods

HTTP request methods indicate the desired action to be performed for a given resource. Different request methods are as follows:

- GET
  The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
  Syntax: `GET /index.html`
  GET requests can be cached and remain in the browser history. They should never be used when dealing with sensitive data.

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

- POST

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. That is, simply put, POST method sends data to the server.
A POST method is typically sent via an HTML form and results in a change on the server. When the POST request is sent via a method other than an HTML form, the body of the request can take any form.
Syntax: POST /test
Example:

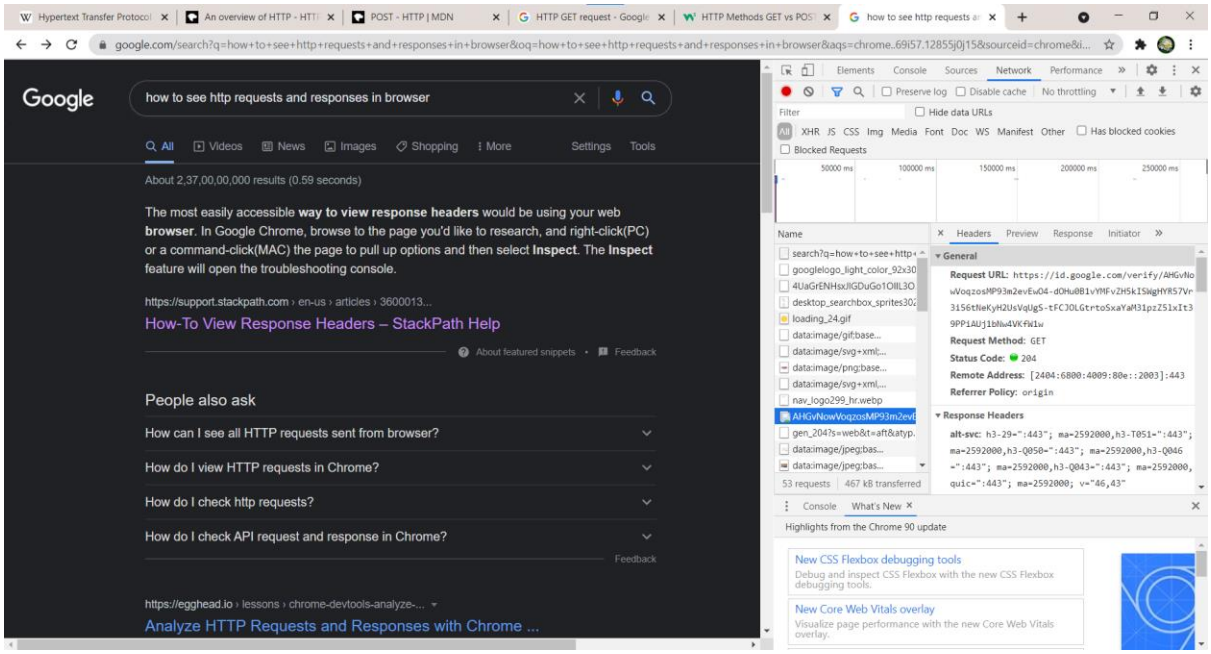
```
POST /test HTTP/1.1
Host: foo.example
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

field1=value1&field2=value2
```

POST requests are never cached and do not remain in browser history. They don't have any restrictions on data length.

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

- DELETE
  The DELETE method deletes the specified resource.
- 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.

Viewing Requests and Responses in a browser



The HTTP requests sent by browser and the responses received from the server can be seen in the network tab of inspect window.

The User Agent header for this particular *request* shown in the screenshot is:

> *Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.85 Safari/537.36*

User Agent

> In computing, a **user agent** is any software, acting on behalf of a user, which "retrieves, renders and facilitates end-user interaction with Web content." A user agent is therefore a special kind of software agent. Some prominent examples of user agents are web browsers and email readers.

Source: User agent - https://en.wikipedia.org/wiki/User_agent

> The **User-Agent** request header is a characteristic string that lets servers and network peers identify the application, operating system, vendor, and/or version of the requesting user agent.

Source: User-Agent - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent

So, User-Agent header is a string which is used to identify the client sending the request. In web browsers, almost all web browsers' user agent string begins with Mozilla/ and has the following general format:

Mozilla/[version] ([system and browser information]) [platform] ([platform details]) [extensions]

The reason for every browser using 'Mozilla' user agent is actually quite sad and funny at the same time. Thank you DevClub for posing this question as I got to know some interesting part of the history of the 'Browser Wars'.

In short, back in '90s, Mosaic used to be the most popular web browser. And then comes Netscape Navigator (which was initially named Mozilla, short for 'Mosaic Killer') with many modern features (like frames, etc.) which Mosaic did not support. It continued to use Mozilla as its User-Agent header. Now, web servers would resort to "User Agent sniffing" to selectively serve content depending on whether the browser is Navigator or not. Now, Microsoft entered the browser game with its Internet Explorer, which like Navigator, supported all modern features at that time. In fact, they were the only two browsers at that time to support them. However, since servers would only serve those "modern" content to only "Mozilla", IE decided to spoof its User Agent to trick servers into believing it is compatible with Navigator and made its User Agent string begin with Mozilla/. Eventually, as it goes, IE dethroned Navigator. But then, the Mozilla group made a comeback with their Gecko engine and Firefox browser. And the Gecko engine was far superior than the IE. And the User Agent sniffing game continued again.

Over time, this practice continued with all upcoming modern browsers (Konqueror (on Linux), then Safari, Opera and finally Chrome) as the web servers are "too lazy" (and in reality, too difficult also now) to update their legacy code.

So, 'Mozilla' here in all these browser's User agent string is just a way to fool web servers and to tell them that they are similar to and compatible with the Gecko engine and can process all the web content as Mozilla's browser.

This article I found describing this in a very good and humorous way:
https://webaim.org/blog/user-agent-string-history/

HTTP Headers

HTTP Headers let the client and server pass additional information with an HTTP request or response. A comprehensive and exhaustive list of HTTP headers can be found here:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

Some of the headers are:

- Host: Specifies the domain name of the server.
- User-Agent: (Described in previous section)
- Server: Contains information about the software used by the origin server.
- Cookie: Contains stored HTTP cookies previously sent by the server.

Cookies

> An HTTP cookie (also called web cookie, Internet cookie, browser cookie, or simply cookie) is a small piece of data stored on the user's computer by the web browser while browsing a website. Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited in the past). They can also be used to remember pieces

> of information that the user previously entered into form fields, such as names, addresses, passwords, and payment card numbers.

Source: HTTP Cookie - https://en.wikipedia.org

Cookies are created to identify the users and personalise their web experience. When a user visits a new website, the web server sends a short stream of information into the web browser containing 'name-value' pairs. The web browser then stores this information locally and when the user visits the same site in future, the browser sends it back to the server as cookies to recall data from the previous session of the user.

Cookies are mainly used for three purposes:

- **Session management:** Logins, shopping carts, game scores, or anything else the server should remember.
- **Personalization:** User preferences, themes, and other settings.
- **Tracking:** Recording and analysing user behaviour.

Creating Cookies:

Cookies are set by the web server by sending the 'Set-Cookie' HTTP Header with the response. The cookies are stored by the browser and then the cookie is sent with the request made to the same server inside the 'Cookie' header. Various attributes can be used with 'Set-Cookie' header to modify various aspects of how cookies are stored and processed.

- **Expires:** specifies the date when the 'Permanent cookies' are deleted.
- **Max-Age:** specifies the duration after which the 'Permanent cookies' are deleted.
- **Secure:** cookies with this attribute are sent to server only with an encrypted request over HTTPS protocol.
- **HttpOnly:** cookies with this attribute are not accessible to JavaScript.
- **Domain:** specifies which hosts are allowed to receive the cookie
- **Path:** specifies the URL path that must exist in the requested URL in order to send the Cookie.
- **SameSite:** lets servers specify whether/when cookies are sent with cross-site requests, which provides some protection against CSRF attacks.

Cookies set by a server hosting the webpage are called "First-party cookies". While, the webpage may contain other components like images, ad banners, etc. which are hosted on other servers. The cookies set by them are called 'Third-party cookies". Third-party cookies let advertisers or analytics companies track an individual's browsing history across the web on any sites that contain their ads. A third-party server can build up a profile of a user's browsing history and habits based on cookies sent to it by the same browser when accessing multiple sites. Consequently, this helps advertisers to track users across different sites and place relevant advertisements.

Cross-Origin Resource Sharing (CORS)

> Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any other origins (domain, scheme, or port) than its own from which a browser should permit loading of resources.

Source: Cross-Origin Resource Sharing - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

In many use cases, resources like public images, videos, scripts, API calls, etc. are needed to be called from a third party website. In such cases, CORS defines a way in which a browser and server can interact to determine whether it is safe to allow the cross-origin request.

The following headers are set for CORS:

- Access-Control-Allow-Origin: Indicates whether the response can be shared.
- Access-Control-Allow-Credentials: Indicates whether the response to the request can be exposed when the credentials flag is true.
- Access-Control-Allow-Headers: Used in response to a preflight request to indicate which HTTP headers can be used when making the actual request.
- Access-Control-Allow-Methods: Specifies the methods allowed when accessing the resource in response to a preflight request.
- Access-Control-Expose-Headers: Indicates which headers can be exposed as part of the response by listing their names.
- Access-Control-Max-Age: Indicates how long the results of a preflight request can be cached.
- Access-Control-Request-Headers: Used when issuing a preflight request to let the server know which HTTP headers will be used when the actual request is made.
- Access-Control-Request-Method: Used when issuing a preflight request to let the server know which HTTP method will be used when the actual request is made.
- Origin: Indicates where a fetch originates from.
- Timing-Allow-Origin: Specifies origins that are allowed to see values of attributes retrieved.

Preflight request:

CORS relies on a mechanism by which browsers make a “preflight” request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request. For "preflighted" requests the browser first sends an HTTP request using the OPTIONS method to the resource on the other origin, in order to determine if the actual request is safe to send. That is, usually, the first request sent is the preflight request.

A sample preflight request:

(*Source: Cross-Origin Resource Sharing - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS*)

```
OPTIONS /doc HTTP/1.1
```

```
Host: bar.other

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101
Firefox/71.0

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

Origin: http://foo.example

Access-Control-Request-Method: POST

Access-Control-Request-Headers: X-PINGOTHER, Content-Type
```