Mangati, Steven Mark Gerald B.

**HTTP (Hypertext Transfer Protocol)**

* The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers.
* A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred
* HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification defines the protocol referred to as “HTTP/1.1”, and is an update to RFC 2068

**URL (Uniform Resource Locator)**

**<scheme>:// <host>: <port>/<path>?<query>#<fragment>**

Example 1:

<http://www.steven.com:80/gp/product/B01723T>

URL Scheme: http

**Host**: [www.steven.com](http://www.steven.com)

**Port**: 80

**URL path**: /gp/product/B01723T

Example 2:

http://www.baguiocity.com:80/burnham.html?q=tourist#place

**Host**: [www.baguiocity.com](http://www.baguiocity.com)

**Port**: 80 (default)

**URL path**: /burnham.html

**Query string**: q=tourist

**Fragment**: #place (referes to the element with id =”place” <div id =place></div>

**HTTP Request and Response**

-A client sends an HTTP request to a server using a message that the server will understand.

-A server responds by sending an HTTP response that the client will understand.

-The request and the response are two different message types.

Request Message Browser Client HTTP server Response Message

**HTTP Request Methods**

HTTP protocol defines a set of request methods. A client can use one of these request methods to send a request message to an HTTP server. The methods are:

* GET: A client can use the GET request to get a web resource from the server.
* HEAD: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.
* POST: Used to post data up to the web server.
* PUT: Ask the server to store the data.
* DELETE: Ask the server to delete the data.
* TRACE: Ask the server to return a diagnostic trace of the actions it takes.
* OPTIONS: Ask the server to return the list of request methods it supports.
* CONNECT: Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.
* Other extension methods.

**“Get” Request Method**

GET is the most common HTTP request method. A client can use the GET request method to request (or "get") for a piece of resource from an HTTP server. A GET request message takes the following syntax

* The keyword GET is case sensitive and must be in uppercase.
* *request-URI*: specifies the path of resource requested, which must begin from the root "/" of the document base directory.
* *HTTP-version*: Either HTTP/1.0 or HTTP/1.1. This client *negotiates* the protocol to be used for the current session. For example, the client may request to use HTTP/1.1. If the server does not support HTTP/1.1, it may inform the client in the response to use HTTP/1.0.
* The client uses the optional request headers (such as Accept, Accept-Language, and etc) to *negotiate* with the server and ask the server to deliver the preferred contents (e.g., in the language that the client preferred).
* GET request message has an optional request body which contains the query string

**Request Header**

A **request header** is an [HTTP header](https://developer.mozilla.org/en-US/docs/Glossary/header) that can be used in an HTTP request, and that doesn't relate to the content of the message. Request headers, like [Accept](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept), [Accept-\*](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language), or [If-\*](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/If-Modified-Since) allow to perform conditional requests; others like [Cookie](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cookie), [User-Agent](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent) or [Referer](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer" \o "The Referer request header contains the address of the previous web page from which a link to the currently requested page was followed. The Referer header allows servers to identify where people are visiting them from and may use that data for analytics, logging, or optimized caching, for example.)precise the context so that the server can tailor the answer.

Not all headers appearing in a request are request headers. For example, the [Content-Length](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Length) appearing in a [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST) request is actually an [entity header](https://developer.mozilla.org/en-US/docs/Glossary/entity_header) referring to the size of the body of the request message. However, these entity headers are often called request headers in such a context.

In addition, [CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS) defines a subset of request headers as [simple headers](https://developer.mozilla.org/en-US/docs/Glossary/simple_header), request headers that are always considered authorized and are not explicitly listed in responses to [preflight](https://developer.mozilla.org/en-US/docs/Glossary/preflight_request) requests.

**\*USING FIDDLER**

**GET /home.html HTTP/1.1**

**Host: developer.mozilla.org**

**User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0**

**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, br**

**Referer: https://developer.mozilla.org/testpage.html**

**Connection: keep-alive**

**Upgrade-Insecure-Requests: 1**

**If-Modified-Since: Mon, 18 Jul 2016 02:36:04 GMT**

**If-None-Match: "c561c68d0ba92bbeb8b0fff2a9199f722e3a621a"**

**Cache-Control: max-age=0**

the [Content-Length](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Length) header in this example is not a request header like the others, but an [entity header](https://developer.mozilla.org/en-US/docs/Glossary/entity_header)

**POST /myform.html HTTP/1.1**

**Host: developer.mozilla.org**

**User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0**

**Content-Length: 128**

**Response Status Code**

The first line of the response message (i.e., the status line) contains the response status code, which is generated by the server to indicate the outcome of the request.

The status code is a 3-digit number:

* 1xx (Informational): Request received, server is continuing the process.
* 2xx (Success): The request was successfully received, understood, accepted and serviced.
* 3xx (Redirection): Further action must be taken in order to complete the request.
* 4xx (Client Error): The request contains bad syntax or cannot be understood.
* 5xx (Server Error): The server failed to fulfill an apparently valid request.

Some commonly encountered status codes are:

* 100 Continue: The server received the request and in the process of giving the response.
* 200 OK: The request is fulfilled.
* 301 Move Permanently: The resource requested for has been permanently moved to a new location. The URL of the new location is given in the response header called Location. The client should issue a new request to the new location. Application should update all references to this new location.
* 302 Found & Redirect (or Move Temporarily): Same as 301, but the new location is temporarily in nature. The client should issue a new request, but applications need not update the references.
* 304 Not Modified: In response to the If-Modified-Since conditional GET request, the server notifies that the resource requested has not been modified.
* 400 Bad Request: Server could not interpret or understand the request, probably syntax error in the request message.
* 401 Authentication Required: The requested resource is protected, and require client’s credential (username/password). The client should re-submit the request with his credential (username/password).
* 403 Forbidden: Server refuses to supply the resource, regardless of identity of client.
* 404 Not Found: The requested resource cannot be found in the server.
* 405 Method Not Allowed: The request method used, e.g., POST, PUT, DELETE, is a valid method. However, the server does not allow that method for the resource requested.
* 408 Request Timeout:
* 414 Request URI too Large:
* 500 Internal Server Error: Server is confused, often caused by an error in the server-side program responding to the request.
* 501 Method Not Implemented: The request method used is invalid (could be caused by a typing error, e.g., "GET" misspell as "Get").
* 502 Bad Gateway: Proxy or Gateway indicates that it receives a bad response from the upstream server.
* 503 Service Unavailable: Server cannot response due to overloading or maintenance. The client can try again later.
* 504 Gateway Timeout: Proxy or Gateway indicates that it receives a timeout from an upstream server.

**Content Negotiation**

HTTP support content negotiation between the client and the server. A client can use additional request headers (such as Accept, Accept-Language, Accept-Charset, Accept-Encoding) to tell the server what it can handle or which content it prefers. If the server possesses multiple versions of the same document in different format, it will return the format that the client prefers. This process is called content negotiation

**Caching**

HTTP caching occurs when the browser stores local copies of web resources for faster retrieval the next time the resource is required. As your application serves resources it can attach cache headers to the response specifying the desired cache behavior

**Cookies**

An HTTP cookie (web cookie, browser cookie) is a small piece of data that a server sends to the user's web browser. The browser may store it and send it back with the next request to the same server. Typically, it's used to tell if two requests came from the same browser — keeping a user logged-in, for example. It remembers stateful information for the [stateless](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#HTTP_is_stateless_but_not_sessionless) HTTP protocol

**SECURITY**

* **Authentication**

[RFC 7235](https://tools.ietf.org/html/rfc7235) defines the HTTP authentication framework which can be used by a server to [challenge](https://developer.mozilla.org/en-US/docs/Glossary/challenge) a client request and by a client to provide authentication information. The challenge and response flow works like this: The server responds to a client with a [401](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/401)(Unauthorized) response status and provides information on how to authorize with a [WWW-Authenticate](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/WWW-Authenticate) response header containing at least one challenge. A client that wants to authenticate itself with a server can then do so by including an [Authorization](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization) request header field with the credentials. Usually a client will present a password prompt to the user and will then issue the request including the correct Authorization header.

Sources:

<https://developer.mozilla.org/en-US/docs/Glossary/Request_header>

<https://devcenter.heroku.com/articles/increasing-application-performance-with-http-cache-headers>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication>

<https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf>

<http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html>

<https://www.slideshare.net/mario_cardinal/http-fundamentals-for-developers>