**SUMMARY OF HTTP:**

Discuss on how HTTP works behind the web browser. Some of the extension of the HTTP which is the WebDav.

**HTTP: Hypertext Transfer Protocol**

* The Hypertext Transfer Control Protocol was invented by Sir Tim Berners-Lee at CERN (Conseil Européen pour la Recherche Nucléaire or European Organization for Nuclear Research)
* It was developed by Word Wide Consortium (W3C) and the Internet Engineering Task Force (IETF).
* It is the standard protocol for transferring data over the World Wide Web (www).
* It is a form of protocol of communication, most especially on how the user communicates with the server.

**SO WHAT IS HTTPS?**

HTTPS or also known as the Hypertext Transfer Protocol Secure, it is a HTTP layered with a security protocol. It is usually used for e-commerce or other secure transactions.

**VERSION HISTORY**

There are a lot of versions of HTTP; it goes from the original HTTP that was defined in 1991 or the HTTP 0.9 until to the recent HTTP 2 that was defined on 2015. But we will focus only to the following HTTP:

|  |  |  |  |
| --- | --- | --- | --- |
|  | HTTP 1.0 (RFC 1945) | HTTP 1.1(RFC 2616) | HTTP 2 (RFC 7540) |
| Month and Year | May 1996 | June 1999 | May 2015 |

**HTTP REQUEST METHOD**

|  |  |
| --- | --- |
| **METHOD** | **Calls server to…** |
| **GET** | Retrieve the resources as the body of the message response that was specified of the Request-URI. |
| **POST** | Pass the body of this request message on as data to be processed by the resource specified by the request-URI |
| **HEAD** | Return the same HTTP header fields that would be returned if a GET method were used, but not return the message body that would be returned to a GET (this provides information about a resource without the communication overhead of transmitting the body of the response, which may be quite large). |
| **OPTIONS** | Return (in Allow header field) a list of HTTP methods that may be used to access the resource specified by the Request-URI |
| **PUT** | Store the body of this message on the server and assign the specified Request-URI to the data stored so that the future GET request messages containing this Request-URI will receive this data in their response message |
| **DELETE** | Respond to future HTTP request messages that contain the specified Request-URI with a response indicating that there is no resource associated with the Request-URI |
| **TRACE** | Return a copy of the complete HTTP request message, including start line, header fields, and body, received by the server. Used primarily for test purposes |

**Reference: Jackson, J. C. (2007). *Web technologies: a computer science perspective (pp 14-15)*. Upper Saddle River, NJ: Pearson/Prentice Hall.**

**SAFE METHOD –** The safe method will only read and should not be expecting any change. It will only fetch the data

**GET**

**HEAD**

**OPTIONS**

**TRACE**

**IDEMPOTENT METHOD –** The effect of the idempotent methods to the responses that there are no changes even if being issued many times.

**GET  
 HEAD**

**OPTIONS**

**TRACE**

**PUT**

**DELETE**

**CACHEABLE METHOD –** Indicates that the response to a method is allowed to be stored for future use. It is also for saving bandwidth.

**GET**

**HEAD**

**POST**

**STATUS CODE DEFINITIONS**

Below is the table of comparisons on the status codes of the HTTP 1.0, HTTP 1.1.

|  |  |  |
| --- | --- | --- |
| **HTTP STATUS CODE DEFINITIONS** | | |
|  | **HTTP 1.0** | **HTTP 1.1** |
| **Informational 1xx** |  | 100 Continue |
|  | 101 Switching Protocol |
| **Successful 2xx** | 200 OK | 200 OK |
| 201 Created | 201 Created |
| 202 Accepted | 202 Accepted |
|  | 203 Non-Authoritative Information |
| 204 No Content | 204 No Content |
|  | 205 Reset Content |
|  | 206 Partial Content |
| **Redirection 3xx** | 300 Multiple Choices | 300 Multiple Choices |
| 301 Moved Permanently | 301 Moved Permanently |
| 302 Moved Temporarily | 302 Found |
|  | 303 See Other |
| 304 Not Modified | 304 Not Modified |
|  | 305 Use Proxy |
|  | 306 (Unused) |
|  | 307 Temporary Redirect |
| **Client Error 4xx** | 400 Bad Request | 400 Bad Request |
| 401 Unauthorized | 401 Unauthorized |
|  | 402 Payment Required |
| 403 Forbidden | 403 Forbidden |
| 404 Not Found | 404 Not Found |
|  | 405 Method Not Allowed |
|  | 406 Not Acceptable |
|  | 407 Proxy Authentication Required |
|  | 408 Request Timeout |
|  | 409 Conflict |
|  | 410 Gone |
|  | 411 Length Required |
|  | 412 Precondition Failed |
|  | 413 Request Entity Too Large |
|  | 414 Request-URI Too Long |
|  | 415 Unsupported Media Type |
|  | 416 Request Range Not Satisfiable |
|  | 417 Expectation Failed |
| **Server Error 5xx** | 500 Internal Server Error | 500 Internal Server Error |
| 501 Not Implemented | 501 Not Implemented |
| 502 Bad Gateway | 502 Bad Gateway |
| 503 Service Unavailable | 503 Service Unavailable |
|  | 504 Gateway Timeout |
|  | 505 HTTP Version Not Supported |

**Informational 1xx**

-Does not define any status in HTTP 1.0 and is not a valid response to the request of HTTP 1.0. It also does not require headers for Informational 1xx. Since in HTTP 1.0 doesn’t define any status codes, clients of HTTP 1.0 must not response to the servers sending 1xx. However in HTTP 1.1, a client of HTTP 1.1 should be prepared to accept one or more status responses of 1xx. User agent may ignore the unexpected status responses.

**Successful 2xx**

– This status code indicates that the request of the client was received, understood and accepted successfully.

**Redirection 3xx**

* This status indicates that there should be an action that needed to be taken to complete a request.

**Client Error 4xx**

* This status indicates that a request that carries bad syntax or cannot be completed

**Server Error 5xx**

* This status indicates that a valid request cannot be fulfill by the server.

**HOW HTTP WORKS?**

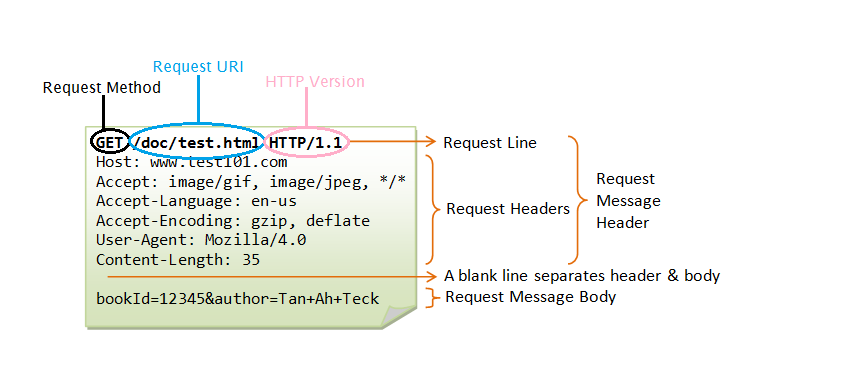
HTTP is a request-response protocol. A good example on how HTTP works is a client opens his Web browser and search for something that is important. When he searches something the web browser is requesting to a server. The request compromises of

* Request line
* Set of request headers
* Request Message body

The server sends a response that comprises

* Status line
* Response headers
* Entity

To understand more on how HTTP works below is an example of a communication between the browser and the server, [www.test101.com](http://www.test101.com). The user requests to go to, [www.test101.com](http://www.test101.com) in response, the browser relay the HTTP request to, [www.test101.com](http://www.test101.com):



**WHAT IS WEBDAV?**