HTTP REQUESTS

A HTTP request consists of between one and three sections. It always starts with a request line. This request line can be followed by zero or more request header lines, and finally, a data section may follow, separated from the headers by an empty line. The data section, should it be present, contains data entered into web forms, including file uploads.

The HTTP request line specifies the HTTP method to use, the path to request from the server, and the version of the HTTP protocol the remainder of the request will use.

The HTTP request headers are specified one per line, as name-value pairs, with the name separated from the value by a **:** character. The headers are used by the client to pass information to the server which it can use in generating its response.

The following is an actual HTTP request by:

```
GET /blog/ HTTP/1.1
Host: www.podfeet.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:37.0) Gecko/20100101
Firefox/37.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en,en-US;q=0.7,ga;q=0.3
```

```
Accept-Encoding: gzip, deflate
Cookie: __utma=188241321.1236907656.1162169166.1408563404.1431184789.53; __qca=P0-1257128144-1331857305112; PHPSESSID=n7uq31arql1uao8g3rahchu743; __utmb=188241321.2.10.1431184789; __utmc=188241321; __utmz=188241321.1431184789.53.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); __utmt=1
DNT: 1
Connection: keep-alive
Cache-Control: max-age=0
```

The first line is the request line, which states that the HTTP **GET** method should be used, that the path **/blog/** is being requested, and that the request is being made using version 1.1 of the HTTP protocol.

The remainder of the lines are request headers, there is no data included in this request. We won't look at all the headers, but I do want to draw attention to a few notable ones.

The **Host** header is what makes it possible for multiple web sites to be served from a single IP address. The receiving web server will have many different domains configured, and this header will tell it which content is being requested.

The **User-Agent** header identifies the browser to the server, and makes it possible to gather browser and OS usage stats.

Notice how you can tell from the above header that I was using FireFox 37 on OS X 10.10.

Notice that any cookies my browser has stored for the domain **podfeet.com** have been added to the request via the **Cookie** header.

Each HTTP request to a server is completely independent of all other requests. There is no relationship between then, no concept of an extended connection or session. This was a major shortcoming of HTTP, and cookies were added later to make it possible for multiple requests to be tied together. When sending a reply to the client, the server can include a **Set-Cookie** header containing a string of text. It is expected that the client will include this cookie in the request headers of all future requests to that same domain until the cookie expires. The server can then tie together all the separate requests into a persistent state, making it possible to log in to websites. Without cookies, there would have been no so-called *web 2.0*!

The **Accept-Language** header enables internationalization of websites. Servers can store multiple versions of the same site in different languages, and use this header to return the correct version to the user.

You might also notice that I have the Do Not Track (**DNT**) header set to 1, which means I am asking not to be tracked.

HTTP Methods

There are quite a few different HTTP methods, but there are only two in common use, **GET** and **POST**.

GET requests should be used when there is little or no form data to send to the server. What little data there may be gets added to the end of the URL after a **?** symbol. **GET** requests should never be used to send sensitive data, as the data is included in the URL, and hence recorded in logs. **GET** requests should be used to retrieve data, and should not be used to alter the internal state of a web app. Because **GET** requests append their data to the end of the URL, and because there is a maximum allowed length for URLs, there is a limit to how much data can be sent using a **GET** request. A big advantage to **GET** requests is that their URLs can be bookmarked and shared with others. E.g., when I use Google to search for something, the text I type into the text box is sent to Google's servers using a **GET** request. I can see it in the URL of the search results. I can then copy and paste that URL into an email to share that search with someone else.

POST requests should be used when there is a lot of data to send, or when the data is sensitive.

POST requests should be used for all request that change the internal state of a web app, e.g. to send an email in a webmail interface, add a post on a social media site, or change a password. **POST** requests add the form data after the headers, so it is not logged, and has no restrictions on the length of the data. **POST** requests cannot be bookmarked or shared.

Encoding Form Data

When ever we submit a web form, the data we have entered is submitted to the server as part f a HTTP request. If the submit button is configured to use **GET**, then the data is appended to the URL, like a Google search, and if the submit button is configured to use **POST**, the data is added to the end of the HTTP request, after the request headers, separated from them by a blank line. However, regardless of how the data is sent, it is always encoded in the same way.

Each form element on a page has a name, and a value. The data is encoded as a sequence of **name=value** pairs, separated with **&** symbols. Neither names nor values can contain special characters, so any such characters in the data must be encoded using URL escape sequences. These are two-digit hexidecimal codes pre-fixed with the **%** symbol. You'll find a full list of URL escape codes here, but as an example, a space character is encode as **%20**.

Source: https://www.bartbusschots.ie/s/2015/05/09/taming-the-terminal-part-34-of-n-introducing-http/