HTTP and Project 2 notes

First connect to the server using a socket on the correct port. The default HTTP port is 443

***HTTP Basics:***

* Like most of the Internet protocols http it is a command and response **text based** protocol using a client server communications model.

Graphical user interface, application

Description automatically generated

* The HTTP protocol is also a stateless protocol meaning that the server isn’t required to store session information, and each request is independent of the other. So…
  + 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.

***Request Structure:***

A request is made up of:

**A command or request + optional headers + optional body content**

Example:

POST /accounts/login/ HTTP/1.1

Host: project2.5700.network

Content-Type: application/x-www-form-urlencoded

Content-Length: 94

Cache-Control: max-age=0

Cookie: csrftoken=Q7MURecDudFRqFwuerAygb0Ek

username=some\_username&password=somepassword&csrfmiddlewaretoken=sometoken&next=%2Ffakebook%2F

NOTE that the content length is the number of characters in the payload.

* Notice that the each header has the following format:

name: value

* Example for the second line in example above
  + name = Host and value = project2.5700.network

***Response Structure:***

The server will respond with a message in the following format;

**A status code + optional headers + optional body content**

Example:

HTTP/1.1 200 OK

Content-Type: text/html; charset=utf-8

Content-Type: text/html; charset=utf-8

Content-Length: 1978

Connection: keep-alive

\<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head> …

***MORE EXAMPLE OF REQUEST AND RESPONSES:***

Diagram

Description automatically generated

***Writing Your own requests:***

* All requests begin with a start line, then have optional header lines and then an optional body/payload.
* Each line of text **MUST end** with a Carriage return (CRLF). These are the “\r\n” characters in ascii

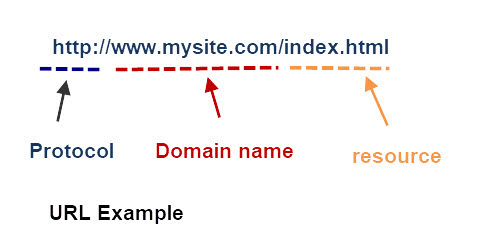
The anatomy of a start line:

Method + Resource Path + Protocol Version

Example:

GET /test.htm HTTP/1.1

* The two main methods/request types are “GET” and “POST”, but there is also “PUT”, “DELETE”… etc
* The resource path is the relative path to the resource and does **NOT** include the domain name.



* HTTP 1.0 and HTTP 1.1 have different requirements and speeds. See <https://www.jmarshall.com/easy/http/> for clarification
* We will be using HTTP 1.1
* The start line is then followed by the optional headers and body

***GET REQUESTS:***

* Get request are used to get data from the server and often do not contain the optional body. An example of a request is below.
* **NOTE** that **CRLF** (carriage returns “\r\n”) are at the end of each line. These are not actual letters, they represent an invisible character, but must be typed explicitly when sending requests.
* **NOTE** the last line ends with an extra newline (**\n**) character. Depends on the site, but this extra new line character signals the end of the whole request, so the server is not waiting for more bytes.

Example of a GET REQUEST sent to server:

GET /testpage.htm HTTP/1.1\r\n

Host: [www.testsite5.com](http://www.testsite5.com)\r\n

Connection: keep-alive\r\n\n

***RESPONSES:***  
The server will then send a response. The first line in the response contains a **status code** and description:

In general there are 5 groups of status codes. Each group has a meaning and is composed of a three digit code:

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

***PROJECT SPECIFIC STATUS CODES***

In terms of the project there are specific codes we need to deal with:

1. 200 OK: Means the request was successful and a correct response will be sent back to the client
2. 302 Found: Means that the resource has been relocated. Make a request to the resource located in the “Location” header
3. 400, 403, 404: Means not found. Request was bad/incorrect format. Also may periodically show up because the server will try to trip up the server sometimes.
4. 503: A server error. Server for some reason cannot send correct response even though request may have been correct. You should resubmit the request at a later time.

Response to GET REQUEST may look like:

HTTP/1.1 200 OK\r\n

Content-Length: 2000\r\n

Connection: Keep-Alive\r\n

Content-Type: text/html\r\n

\r\n

\n

\n

\n

<html> …. <html>\r\n\n

* If there is a “Content-Length” Header then the server will send some payload/data after sending the optional headers. It often tells you the amount of data sent to the client.
* Note the extra CRLF (\r\n) and three new line characters (\n) . These often will act as a delimiter between the response and the payload. This delimiter **MAY DIFFER** between sites.

POST REQUEST:

* post requests are often used to send data to the server, i.e. login etc.
* These post request need a “**Content-Length**” header with the length of the payload to send.
  + The content-length value **MUST** match the amount of char in the payload to send.
* The most often method of sending a payload is to use **urlencoding**.
* Need to specify this in the **“Content-Type”** header.
* Data payload is in the format of key: value. The text payload must be written in the format of

key1=value1&key2=value2

Example:

POST /accounts/login/ HTTP/1.1\r\n

Host: project2.5700.network\r\n

Content-Type: application/x-www-form-urlencoded\r\n

Content-Length: 94\r\n

Cache-Control: max-age=0\r\n

Cookie: csrftoken=Q7MURecDudFRqFwuerAygb0Ek\r\n

\r\n

username=some\_username&password=somepassword&csrfmiddlewaretoken=sometoken&next=%2Ffakebook%2F\r\n\n

***ProTip See it in real time on your own browser:***

* If you want to see http requests made by your browser (chrome or Microsoft edge) you can! It’s super useful for learning http:
  + right click on any website -> inspect
* This will bring you to this window, select “Network” from the top tab. Optionally you can select “Preserve Log” to see any and all requests made. See figure below.

Text

Description automatically generated

* Select an interaction on the left hand tab:
* You can click “View Source” which will show the raw request/response instead of the parsed version
* **Please NOTE**: only some interactions allow you to switch between “View parsed” and “View source”

Text

Description automatically generated

* Firefox can also show http requests with the http header live tool

Citation:

http://www.steves-internet-guide.com/http-basics/

***PROJECT SPECIFIC NOTES***

1. Send a GET request to the resource “/accounts/login/” to obtain a response with an html payload
   1. This html payload contains the cookie/csrftoken
   2. Save this token
2. Send a POST request to “/accounts/login/”
   1. This POST request must use the csrftoken obtain in step 1
   2. The payload must contain the following keys: username, password, csrfmiddlewaretoken, and next
   3. The response from this request may sometime response with a failure even with a correct username and password, because the server is trying to trip up the crawler. So just run the request again
3. Parse the response from the POST request from step 2.
   1. This response contains 2 new cookies (a new csrftoken and a sessid)
   2. These 2 cookies will be used for all future requests
   3. The response also contains a body filled with links
4. Crawl through the links (GET request the links) and parse them along the way.
   1. Look for a secret flag at each GET request
   2. Add more links to visit (if not visited) to the crawlers to visit list
   3. handle the response codes appropriately.
   4. Sometimes the server will respond with “dud”/incomplete htmls. Ignore these and continue.
   5. Sometimes the server not respond at all, disconnect the current socket and reconnect with a new socket.
   6. Sometimes the sever will drop the connection. Disconnect the current socket and reconnect with a new socket.
5. Stop crawling once 5 flags have been found.