Name: Aaron Neagle Date: 1/13/2023 \_

**Overview**

HTTP is the de facto standard for data-interchange and document-transfer on the internet. Web developers require a deep knowledge of the specification to understand how browsers interpret documents on a website. As well, learning HTTP helps a web developer understand how to build software systems that communicate across a network. This lab will highlight the most important principles of HTTP.

**How to Complete & Submit Your Work**

Complete the lab steps below and provide answers for each question and evidence of your work as necessary. If you are working as a group, each group member should submit their own individual answers. Once complete, upload this document to Canvas.

**URI Formats**

URIs (Uniform Resource Indicators) are used to identify resources on the web. Since web addresses also provide an access mechanism, they can be more accurately described as URLs (Uniform Resource Locators). For this lab, we will prefer the term URI.

Each URI on the web is broken up into several parts:

scheme://authority@host:port/path?query#fragment

* **Scheme** – This is protocol to access the web resource. This will be HTTP for most of our conversations this quarter.
* **Authority** – Credentials that are provided to access secure resources.
* **Host** – The domain (or IP address) of the host machine.
* **Port** – The port on the remote server to communicate on.
* **Path** – The physical or logical path to the resource on the host system.
* **Query** – A query string that can be used to parameterize the URI.
* **Fragment** – Identifies a part of a resource on the web. This is usually used for internal page references for document URIs.

Complete the table below by identifying each individual part of the provided URI.

|  |  |
| --- | --- |
| http://www.msn.com/news | Scheme: http:// |
| Authority: |
| Host: [www.msn.com](http://www.msn.com) |
| Port: |
| Path: /news |
| Query: |
| Fragment: |

|  |  |
| --- | --- |
| http://www.surplus.com/viewcart.php?custid=1020&err=0 | Scheme: http:// |
| Authority: |
| Host: [www.surplus.com](http://www.surplus.com) |
| Port: |
| Path: /viewcart.php |
| Query: custid=1020&err=0 |
| Fragment: |

|  |  |
| --- | --- |
| ftp://washington.state.gov/viewrules.html#rule3 | Scheme: ftp:// |
| Authority: |
| Host: Washington.state.gov |
| Port: |
| Path: /viewrules.html |
| Query: |
| Fragment: #rule3 |

|  |  |
| --- | --- |
| http://localhost:8000/test.php?debug=1 | Scheme: http:// |
| Authority: |
| Host: localhost |
| Port: :8000 |
| Path: /test.php |
| Query: ?debug=1 |
| Fragment: |

Take a moment to use your browser and visit some of the websites that you normally frequent. As you visit each page, pay attention to the web addresses listed in your browser. Provide an actual example of an URI of the following types that you found:

A URI with a non-empty path: https://egator.greenriver.edu/courses/2308756/assignments/28063110

A URI with a query string: https://www.linkedin.com/notifications/?filter=all

Using a query string to parameterize web requests can be used in many ways. Some common uses are: passing form values to a server, personalizing web pages, or tracking analytics on a website. Query strings are broken up into key/value pairs. A query string beings with the question mark (?). Each key/value pair is declared with an equals (=) symbol and separated by ampersands (&).

Take the URI below and identify all keys and values in the query string.

|  |  |  |
| --- | --- | --- |
| **URI** | **Keys** | **Values** |
| https://seattle.market.com/produce?page=local&campaign=fall&prodid=1019&sale=true | page | Local |
| Campaign | fall |
| Prodid | 1019 |
| Sale | true |
|  |  |
|  |  |

**HTTP Requests**

Diagram

Description automatically generated

The transfer of documents (or data) on the web follows a request-response cycle. An **HTTP request** is sent to the server with enough information to identify a resource and how to access it. The server then responds with an **HTTP response**.

HTTP requests have several parts:

* A request line (the first line of the request)
* A list of HTTP headers that define how to make the request
* A space
* A request body, which can hold general text content sent with the request to the server. Not every HTTP request has a body.

Text

Description automatically generated

The following information can be found in the request line and headers:

* The location of the web server
* The version of HTTP used in the request
* The path to the resource requested
* An HTTP verb
* The expected format of the resource
* Instructions on how to cache the request

Consider the example HTTP request seen below. This request was sent when asking for "news.google.com" in the browser.

GET /topstories?hl=en-US&gl=US&ceid=US:en HTTP/1.1

Host: news.google.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/93.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate, br

Alt-Used: news.google.com

Connection: keep-alive

Cookie: 1P\_JAR=2021-10-17-18;

SEARCH\_SAMESITE=CgQI5ZMB; OGPC=19025377-1:19025836-2:19026101-2:; SID=DAjYG2hcCuVDzodUZM5SKb-vV94xfkvzsUE6QkR8vRtsVoFnuDdreeTU2sZ4g493uN1Dyw.;

Upgrade-Insecure-Requests: 1

Sec-Fetch-Dest: document

Sec-Fetch-Mode: navigate

Sec-Fetch-Site: none

Sec-Fetch-User: ?1

Pragma: no-cache

Cache-Control: no-cache

Fill in this table by identifying each part of the example HTTP request.

|  |  |
| --- | --- |
| **HTTP Request Attribute** | **Value** |
| The location of the web server | News.google.com |
| The version of HTTP used in the request | HTTP/1.1 |
| The path to the resource requested | topstories |
| An HTTP verb | GET |
| The expected format of the resource | text |
| Instructions on how to cache the request | No-cache |

One of the difficulties of communications over HTTP is that transmissions are entirely text-based. Since there are multiple different file formats used on the web, it can be difficult to identify what type of text is being sent across a network. For example, web developers regularly use the following text formats:

* HTML – structured web documents
* Javascript – client side programming logic
* CSS – style definitions
* JSON – well-formatted data records

MIME types are the mechanism that is used to signal what type of text is being sent over network communications. Each request or response will include a MIME value of the form: type/subtype.

Here are some of the MIME types you are likely to encounter as a web developer:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **.bmp** | image/bmp | **.jpg** | image/jpeg | **.svg** | image/svg+xml |
| **.css** | text/css | **.js** | application/x-javascript | **.swf** | application/x-shockwave-flash |
| **.dtd** | application/xml-dtd | **.json** | application/json | **.tiff** | image/tiff |
| **.doc** | application/msword | **.mp3** | audio/mpeg | **.txt** | text/plain |
| **.exe** | application/octet-stream | **.mpeg** | video/mpeg | **.wav** | audio/wav, audio/x-wav |
| **.gif** | image/gif | **.pdf** | application/pdf | **.xls** | application/vnd.ms-excel |
| **.html** | text/html | **.png** | image/png | **.xml** | application/xml |
| **.jar** | application/java-archive | **.ppt** | application/vnd.ms-powerpoint | **.zip** | application/zip, application/x-compressed-zip |

Take a second look at the example HTTP request for news.google.com listed previously. Identify at least five MIME types in the request.

1. text/html

2. application/xhtml+xml

3. application/xml

4. image/avif

5. image/webp

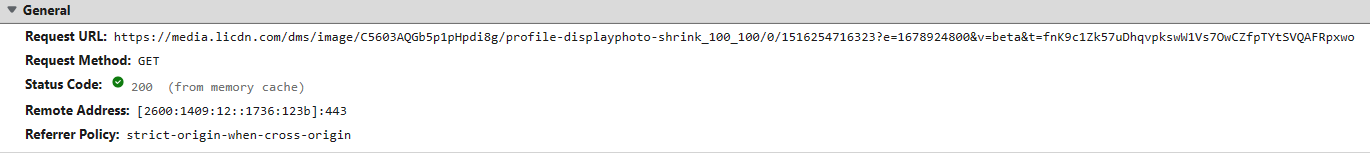
Let's explore how to view HTTP requests in your browser. Open your favorite browser and visit a web page you're familiar with. Then open the inspector in the browser and search for network tools. Using these tools, identify where the browser shows you the contents of an HTTP request.

Please note the following:

* Each browser inspector is slightly different, so you will need to experiment to see how to view network traffic.
* Each browser does give you access to HTTP requests/responses. Keep looking if you're not seeing this information right away.

Once you have found the HTTP request for the page you visited, take a screenshot of both the headers and the request body (the HTML). If possible, you should display the raw text that the browser provides, rather than formatted tables or HTML.

Place screenshot here...



**HTTP Responses**

HTTP responses are similar to HTTP requests in that they share the same common elements:

* A response line (the first line of the response)
* A list of HTTP headers that define how the server responds
* A space
* A response body, which can hold general text content sent with the response. Not every HTTP response has a body, but it is common to include text content here. For example, when loading a web page over HTTP the HTML for the page is included in the response body.

Text, table

Description automatically generated

The following information can be found in the response line or headers:

* The type of web server
* The version of HTTP used in the request
* An HTTP status code
* The format of the content sent back with the response
* The date the resource was accessed or last modified
* The number of bytes in the response body

Consider the example HTTP response seen below. This response was received when asking for "news.google.com" in the browser.

HTTP/3 200 OK

content-type: text/html; charset=utf-8

vary: Sec-Fetch-Dest, Sec-Fetch-Mode, Sec-Fetch-Site

x-ua-compatible: IE=edge

cache-control: no-cache, no-store, max-age=0, must-revalidate

pragma: no-cache

expires: Mon, 01 Jan 1990 00:00:00 GMT

date: Sun, 17 Oct 2021 20:39:43 GMT

strict-transport-security: max-age=31536000

content-security-policy: require-trusted-types-for 'script';

cross-origin-resource-policy: same-site

cross-origin-opener-policy: unsafe-none

content-encoding: gzip

server: ESF

x-xss-protection: 0

x-frame-options: SAMEORIGIN

x-content-type-options: nosniff

set-cookie: SIDCC=AJi4QfFB2WStcUDWtgMPLNQcZb4tGd8cOOfiiqVAJQ0HFyz0\_D\_-yWuppnZMRKSgENax8kXuPj3b; expires=Mon, 17-Oct-2022 20:39:43 GMT; path=/; domain=.google.com; priority=high

alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-T051=":443"; ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443"; ma=2592000; v="46,43"

<!doctype html>

<html lang="en" dir="ltr"><head>

<base href="https://news.google.com/">

<meta name="referrer" content="origin">

<link rel="canonical" href="https://news.google.com/topstories">

<meta name="viewport" content="width=device-width,initial-scale=1,minimal-ui">

<meta name="google-site-verification"

content="AcBy5YFny2HQgVUCR18tO5YUTf6MpVlcJqGTd-a9-SI">

...

</html>

Consider the header fields listed below. Explain how these headers are being used in the example HTTP response for news.google.com.

content-type: describes the type of information being returned. text/html

cache-control: describes if information is being cached or requires revalidation

One of the more important parts of an HTTP response is the status code. A status code is provided that indicates how the server handled the request. This can be a message that is informational, a success message, a redirection, or an error message.

Consider this (partial) table of HTTP status codes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1×× Informational** | **2×× Success** | **3×× Redirection** | **4×× Client Error** | **5×× Server Error** |
| 100 Continue  101 Switching Protocols  102 Processing | 200 OK  201 Created  202 Accepted  203 Non-authoritative Information  204 No Content  205 Reset Content  206 Partial Content  207 Multi-Status  208 Already Reported  226 IM Used | 300 Multiple Choices  301 Moved Permanently  302 Found  303 See Other  304 Not Modified  305 Use Proxy  307 Temporary Redirect  308 Permanent Redirect | 400 Bad Request  401 Unauthorized  402 Payment Required  403 Forbidden  404 Not Found  405 Method Not Allowed  406 Not Acceptable  407 Proxy Authentication Required  408 Request Timeout  409 Conflict  410 Gone  411 Length Required  ... | 500 Internal Server Error  501 Not Implemented  502 Bad Gateway  503 Service Unavailable  504 Gateway Timeout  505 HTTP Version Not Supported  506 Variant Also Negotiates  507 Insufficient Storage  508 Loop Detected  510 Not Extended  511 Network Authentication Required  599 Network Connect Timeout Error |

What HTTP response code was used in the HTTP response above for news.google.com?

200 OK

Let's explore how to view HTTP responses in your browser. Perform the same steps you previously completed to view the network information in your browser.

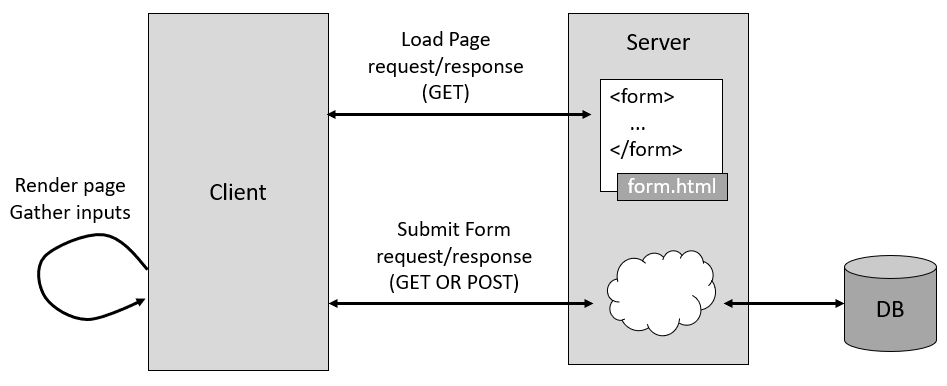
Pick a page to visit and identify the response headers and response body for your request. Take a screenshot of both and place them below. If possible, you should display the raw text that the browser provides, rather than formatted tables or HTML.

Place screenshot here...



**Further Practice**

Let's try a familiar routine for interacting with users and data in the browser. You have already worked with HTML forms and how to handle user inputs. The HTTP verbs used from interactions follow this diagram:



An application with a form has been provided for you. Download this code (<https://github.com/joshbarcher/SDEV372-Http-Lab>), unzip it, load it into your editor, and start the project. You should be able to then visit a simple form at <http://localhost:8080/registration/add>.

Open the network tab in your browser inspector, then enter (valid) data values into the form and submit it. Take a look at the HTTP request/response during this interaction. Mark anything different from our previous examples, or anything interesting you see, in the area below:

There were initially no network requests. Once form was submitted the requests were generated.

Lastly, use the network tool in your browser to find the form's key/value pairs that are sent from the browser to the server. Take a screenshot of these key/value pairs and place it below.

Place screenshot here...

