**How Web Works Exercise**

**Part One: Solidify Terminology**

In your own terms, define the following terms:

* What is HTTP?

*// Hypertext Transfer Protocol – is a set of rules of how to communicate between a browser and a web server.*

* What is a URL?

// Universal Resource Locators - Commonly used to reference webpages (http requests). Includes the protocol (http), Hostname (nickname for the server, i.e. google.com), Port (what the server talks to), Resource (tells server what information to respond with) and Query String (specifies different parameters (key/value pairs) of information within the resource and on the server that you want for the output)

* What is DNS?

// Domain Name Service - Is like the phonebook for the web, assigning IP addresses to a Hostnames and vice versa.

* What is a query string?

// Allows different parameters (key/value pairs) of information to be passed into the URL in the format ?key1=value1&key2=value2 … that you want to be included in the response.

* What are two HTTP verbs and how are they different?

// GET requests data from the server without side effects (i.e., search forms, web pages)

//POST sends data to the server with side effects (i.e., pages that change data on the server)

* What is an HTTP request?

//When a web user points their browser to a webpage on the server (i.e., hitting enter after entering URL in nav bar), the browser makes a request to that server following the HTTP protocol. This is almost always a GET request. For example, a request for HTML from news.google.com)

* What is an HTTP response?

//The server receives some information from a request, it processes that information and determines the appropriate response to the client following the HTTP protocol (i.e. sending back HTML/CSS/JS, etc)

* What is an HTTP header? Give a couple examples of request and response headers you have seen.

// Request headers contain requested information we want in a response (i.e., hostname you’re asking about, date the browser thinks it is, language the browser wants info in). Response headers contain information about the response we get back (i.e., date/time server thinks it is, last modifie\content type(html), cookies server wants to set)

* What are the processes that happen when you type “<http://somesite.com/some/page.html>” into a browser?

//The DNS converts the hostname into an IP address (unless it’s already in the computer cache, the router or the internet service provider has it), the browser connects to the IP address on the default port (80). Using HTTP protocol, the resource /some/page.html is asked for. The response also follows HTTP protocol and includes a response body (the data/content being sent back to the browser typically HTML, CSS and JS and headers).

From Solution:

1. Your browser “resolves” the name into an IP address using DNS
2. Your browser makes a request to that IP address, including headers (info about browser, any previous cookies, and other things)
3. The server sends a response (typically, HTML, with a status code (200 if it was successful)
4. The browser makes a DOM from that HTML, and finds any other resources needed (images, CSS, JavaScript, etc.)
5. The browser makes separate HTTP requests for those resources and receives response from the server for each

**Part Two: Practice Tools**

1. Using ***curl***, make a ***GET*** request to the *icanhazdadjoke.com* API to find all jokes involving the word “pirate”

// curl https://icanhazdadjoke.com/search?term=pirate

// Why couldn't the kid see the pirate movie? Because it was rated arrr!

//What does a pirate pay for his corn? A buccaneer!

//What did the pirate say on his 80th birthday? Aye Matey!

//Why are pirates called pirates? Because they arrr!

//Why do pirates not know the alphabet? They always get stuck at "C"

1. Use ***dig*** to find what the IP address is for *icanhazdadjoke.com*

*//*172.67.198.173

1. Make a simple web page and serve it using ***python3 -m http.server***. Visit the page in a browser.

// type $ python3 -m http.server in terminal

// type localhost:8000 in browser to get a list of all the files in that directory. Click on file you want to see like the index.html file.

## Part Three: Explore Dev Tools

Build a very simple HTML form that uses the GET method (it can use the same page URL for the action) when the form is submitted.

// <form action="/search" method="GET">

      <input type="search" name="q" />

      <button type="Search">Search</button>

    </form>

Add a field or two to the form and, after submitting it, explore in Chrome Developer tools how you can view the request and response headers.

// Query string is added with nav bar changing to: file:///C:/search?q=pickles

// Open Dev Tools from the html file, click on the Network tab at the top. Then submit something from the form in the browser and look in Dev Tools. Click on one of the files or images or search results, etc. listed on the left under Name. This will bring up some General information, the Request, Response, Headers, etc.

Edit the page to change the form type to POST, refresh in the browser and re-submit. Do you still see the field in the query string?

//No. Nav bar shows <file:///C:/search>

<form action="/search" method="POST">

      <input type="search" name="q" />

      <button type="Search">Search</button>

    </form>

Explore in Chrome how you can view the request and response headers, as well as the form data.

## Part Four: Explore the URL API

At times, it’s useful for your JavaScript to look at the URL of the browser window and change how the script works depending on parts of that (particularly the query string).

[Read about the URL API](https://developer.mozilla.org/en-US/docs/Web/API/URL)

Try some of the code examples in the Chrome Console so that you can get comfortable with the basic methods and properties for instances of the URL class.

The **URL** interface is used to parse, construct, normalize, and encode [URLs](https://developer.mozilla.org/en-US/docs/Glossary/URL). It works by providing properties which allow you to easily read and modify the components of a URL.

You normally create a new URL object by specifying the URL as a string when calling its constructor, or by providing a relative URL and a base URL. You can then easily read the parsed components of the URL or make changes to the URL.

[**Constructor**](https://developer.mozilla.org/en-US/docs/Web/API/URL#constructor)

// [URL()](https://developer.mozilla.org/en-US/docs/Web/API/URL/URL) - Creates and returns a URL object referencing the URL specified using an absolute URL string, or a relative URL string and a base URL string.

// The constructor takes a url parameter (pathname), and an optional base parameter to use as a base if the url parameter is a relative URL.

//hostname: A string containing the domain of the URL.

//pathname - A string containing an initial '/' followed by the path of the URL, not including the query string or fragment.

const url = new URL('../cats', 'http://www.example.com/dogs');

console.log(url.hostname); // "www.example.com"

console.log(url.pathname); // "/cats"

URL properties can be set to construct the URL:

// hash- A string containing a '#' followed by the fragment identifier of the URL.

//href - A [stringifier](https://developer.mozilla.org/en-US/docs/Glossary/Stringifier) that returns a string containing the whole URL.

url.hash = 'tabby';

console.log(url.href); // "http://www.example.com/cats#tabby"

URLs are encoded according to the rules found in [RFC 3986](https://datatracker.ietf.org/doc/html/rfc3986). For instance:

// pathname: A string containing an initial '/' followed by the path of the URL, not including the query string or fragment.

//href - A [stringifier](https://developer.mozilla.org/en-US/docs/Glossary/Stringifier) that returns a string containing the whole URL.

url.pathname = 'démonstration.html';

console.log(url.href); // “<http://www.example.com/d%C3%A9monstration.html>”

The [URLSearchParams](https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams) interface can be used to build and manipulate the URL query string.

To get the search params from the current window's URL, you can do this:

// searchParams - A [URLSearchParams](https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams) object which can be used to access the individual query parameters found in search.

// https://some.site/?id=123

const parsedUrl = new URL(window.location.href);

console.log(parsedUrl.searchParams.get("id")); // "123"

The [toString()](https://developer.mozilla.org/en-US/docs/Web/API/URL/toString) method of URL just returns the value of the [href](https://developer.mozilla.org/en-US/docs/Web/API/URL/href) property, so the constructor can be used to normalize and encode a URL directly.

//[toString()](https://developer.mozilla.org/en-US/docs/Web/API/URL/toString) - Returns a string containing the whole URL. It is a synonym for [URL.href](https://developer.mozilla.org/en-US/docs/Web/API/URL/href), though it can't be used to modify the value.

const response = await fetch(new URL('http://www.example.com/démonstration.html'));