How the Web Works

In this lab, you'll be working with a partner to explore a little more about the internet, the web, requests, responses and more. You'll be reading and writing about concepts as well as practicing some of the commands that we saw during the lecture earlier.

Topic 1: The Internet and the World Wide Web

1) What is the internet? (hint: here)
   a global computer network providing a variety of information and communication facilities.

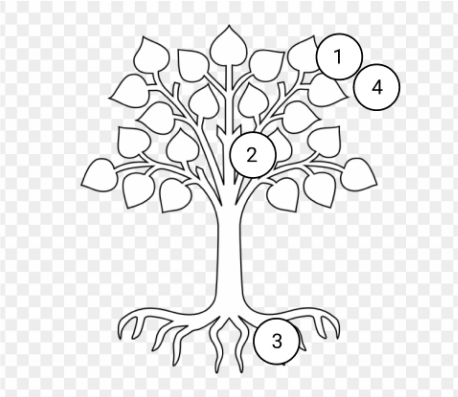
2) What is the world wide web? (hint: here)
   an information system on the internet which allows documents to be connected to other documents by hypertext links.

3) Partner One: read this page on how the internet works, Partner Two: read this page on how the world wide web works. When you're done reading, come back together and and answer the following questions
   a) What are networks?
      Networks are connections between 2 or more computers, servers or routers.
   b) What are servers?
      Servers are computers that store web pages, sites or apps. It can provide functionality for other computers or "clients".
   c) What are routers?
      A **router** is a networking device that forwards data packets between computer networks
   d) What are packets?
      Packets are small chunks of data that make it easier to replace when the connection drops or becomes corrupted.

4) Come up with a metaphor for the internet and the web, you can do a single one if you think of one that puts them together or two separate ones (feel free to use one you've heard today or read about if you can't think of a new one, but spend at least 10 minutes trying to think of something different before you resort to that)

   The internet is **like a tree**. The last server that serves the website is like a root, and we query all the leaves to the branches to the trunk to the roots to find the base website.

5) Draw out a diagram of the infrastructure of the internet and how a request and response travel using your metaphor (like the map and letters we saw during the lecture). Insert the drawing into this document (can be a picture of a physical drawing, a Google Drawing, a Figma drawing, etc)



1. The browser goes to the DNS server, and finds the real address of the server that the website lives on (you find the address of the shop).
2. The browser sends an HTTP request message to the server, asking it to send a copy of the website to the client (you go to the shop and order your goods). This message, and all other data sent between the client and the server, is sent across your internet connection using TCP/IP.
3. If the server approves the client's request, the server sends the client a "200 OK" message, which means "Of course you can look at that website! Here it is", and then starts sending the website's files to the browser as a series of small chunks called data packets (the shop gives you your goods, and you bring them back to your house).
4. The browser assembles the small chunks into a complete web page and displays it to you (the goods arrive at your door — new shiny stuff, awesome!).

Topic 2: IP Addresses and Domains

1) What is the difference between an IP address and a domain name?

   The **IP address** is an actual set of numerical instructions. It communicates exact information about the address in a way that is useful to the computer but makes no sense to humans. The **domain name** functions as a human-readable link to the IP address.

2) What's devmountain.com's IP address? (Hint: use 'ping' in the terminal)

   104.22.13.35 (andrew)
   172.67.9.59 (Levi)

3) Try to access devmountain.com by its IP address. It shouldn't work because we have our sites protected by a service called CloudFlare. Why might it be important to not let users access your site directly at the IP address?

   Error: 1003
   It's important to prevent hackers.

4) How do our browsers know the IP address of a website when we type in its domain name? (If you need a refresher, go read this comic linked in the handout from this lecture)

   The browser knows the IP address through a **DNS (domain name server)** which keeps track of IP addresses and their corresponding domain name.

Topic 3: How a web page loads into a browser

The steps of how a web page is requested and sent are in the table below. However, **they are out of order**. Unscramble them and explain your thinking/reasoning in the second two columns of the table.

Steps Scrambled	Steps in Correct Order	Why did you put this step in this position?
Example: Here is an example step	*Here is an example step*	*- I put this step first because ____*

		- I put this step before/after ____ because ____
Request reaches app server	Initial request (link clicked, URL visited)	This would be the initial Get request.
HTML processing finishes	Request reaches app server	This would be after the server finishes the Get request and returns the HTML file.
App code finishes execution	App code finishes execution	I put this step here because the server needs to process the request before sending the relevant data back to the client.
Initial request (link clicked, URL visited)	Browser receives HTML, begins	I put this step after app code because the client has received the data from the server at this point.
Page rendered in browser	HTML processing finishes	I put this step second to last because the browser needs to process HTML before rendering it with the browser engine.
Browser receives HTML, begins processing	Page rendered in browser	I put this step last because when the browser receives all the data it can render the page.

Topic 4: Requests and Responses

Setup

- Download the folder for this exercise from Frodo.
- Make sure you unzip it.
- Open it in VS Code
- Run `npm i` in the terminal (make sure you're in the web-works folder you just downloaded).
    - You'll know it was successful if you see a node_modules folder in the web-works folder.
- Run `node server.js` in the terminal (also in the web-works folder) and you should see a log to the terminal saying 'serving up port 4500'
- You'll be using this file to figure out what will happen when you make requests to this server, so read it over to see what's going on. We'll be getting into the two GET functions and the POST function.

Part A: GET /

- You'll start by looking at the function that runs when we make a get request to /, which looks like this: http://localhost:4500 or http://localhost:4500/
- You'll use the curl command to make a request and read the response in your terminal

1) Predict what you'll see as the body of the response:

   H1 tag which is the response for querying the '/' index with the GET HTTP method.

2) Predict what the content-type of the response will be:

- Open a terminal window and run `curl -i http:localhost:4500`

Should show information about the GET response from the server, as well as information about HTTP response headers (with the -i command)

3) Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why?

Yes we predicted the body would be the same as accessing it using a browser GET request with the url http://localhost:4500. However we did not predict the exact response headers it would return including the server info (powered by Express), date, connection information, and content-type (text/html, utf-8).

4) Were you correct about the content-type of the response? If yes, how/why did you make your prediction? If not, what was it and why?

Yes we thought the content-type would be text/html because the code of server.js says it will return <h1>s and <h2>s encapsulating some text data.

Part B: GET /entries

- Now look at the next function, the one that runs on get requests to /entries.
- You'll use the curl command again. This time, you'll need to figure out how to modify it to get the response that you need.

1) Predict what you'll see as the body of the response:

When we type curl -i localhost:4500/entries we predict to see the object with all the entries.

2) Predict what the content-type of the response will be:

JSON object ?

- In your terminal, run a curl command to get request this server for /entries

3) Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why?
Yes we were correct as we predicted the server would return a Javascript object with all the entries in it.

4) Were you correct about the content-type of the response? If yes, how/why did you make your prediction? If not, what was it and why?

Almost correct - it's application/json.

Part C: POST /entry

- Last, read over the function that runs a post request.

1) At a base level, what is this function doing? (There are four parts to this)

a) First it creates a newEntry and populates it with data including id, date, and content.
b) Then it pushes data into the entries array
c) Following which, it increments the globalId counter.
d) Finally it sends the updated entries object back to the client.

2) To get this function to work, we need to send a body object with our request. Looking at the function in server.js, what properties do you know you'll need to include on that body object? And what data types will they be (hint: look at the objects in the entries array)?

   We need to include a date string and a content string to help with our request, to create a newEntry object.

3) Plan the object that you'll send with your request. Remember that it needs to be written as a JSON object inside strings. JSON objects properties/keys and values need to be in **double quotes** and separated by commas.
   {
   "date": "June 21",
   "content": "content"
   }

4) What URL will you be making this request to?

   http://localhost:4500/entry with a POST request

5) Predict what you'll see as the body of the response:

   We predict we'll see the entries object returned to us.

6) Predict what the content-type of the response will be:

   We predict the content-type to be: application/JSON.

- In your terminal, enter the curl command to make this request. It should look something like the example below, with the information you decided on in steps 3 and 4 instead of the ALL CAPS WORDS.
  - curl -i -X POST -H 'Content-type: application/json' -d JSONOBJECT URL
  - curl -i -X -POST -H "Content-type: application/json" -d '{"date":"today", "content":"123" }' http://localhost:4500/entry

7) Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why?

   Yes. it returned the JSON object with the new entry.

8) Were you correct about the content-type of the response? If yes, how/why did you make your prediction? If not, what was it and why?

   Yes it was an application/JSON format returned to us.

Submission

1. Save this document as a PDF
2. Go to Github and create a new repository. (Click the little + in the upper right hand corner.)
3. Name your repository "web-works" (or something like that).
4. Click "uploading an existing file" under the "Quick setup heading".

5. Choose your web works PDF document to upload.
6. Add “commit message” under the heading “Commit changes”. A good commit message would be something like “Adding web works problems.”
7. Click commit changes.

Further Study: More curl

Visit this link and do the exercises using the website provided. Keep track of the commands you used in this document. (Don’t forget to resubmit to GitHub when you complete this section)