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)
    - World wide network of networks that uses the internet protocol suite.
      A large network of computers that communicate with each other.
2) What is the world wide web? (hint: here)
    - Interconnected system of public webpages accessible through the internet.
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?
        - A system that allows computers, devices to communicate with each other.
    b) What are servers?
        - Computers that store webpages, sites, or apps.
    c) What are routers?
        - A small computer that ensures that the a message sent from some computer arrives at the correct destination
    d) What are packets?
        - Packets are small pieces of data that make up the website you are trying to download from. They are broken down into small pieces so many people can download at the same time.
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)
    - A train railway system. The train, railway system that takes you places is the internet itself. The web would be the different locations the train can take you to.
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)

Topic 2: IP Addresses and Domains

1) What is the difference between an IP address and a domain name?
    - An IP address is a unique address of a computer that is linked to a network. (in the form of four numbers (0-255) connected by dots.)
    - A domain name is a unique name in the form of a string associated with the IP address that allows users to memorize and communicate with the address easily.
2) What's devmountain.com's IP address? (Hint: use 'ping' in the terminal)
    - 2606:4700:10::6816:d23

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?
    - Giving away IP address can lead to malicious attacks on the website.

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)
    - When the user types in the domain name, the domain name service (DNS) converts the domain name into the IP address, which the computer then uses to navigate to the correct server.

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	I put this step first because in order for a server to receive a request, there needs to be request made by some local host.
HTML processing finishes	Request reaches app server	I put this step before "App code finishes execution" because in order for the App code to execute, the request needs to reach the app server.
App code finishes execution	App code finishes execution	Once the server receive a request, the app code executes and finishes.
Initial request (link clicked, URL visited)	Browser receives HTML, begins processing	In order for HTML to process, the browser needs to receive related information from the app server.
Page rendered in browser	HTML processing finishes	Once the browser receives HTML in formation, it processes before rendering it,
Browser receives HTML, begins processing	Page rendered in browser	The last step.

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:
   Jurrni. Journaling your journeys.
2) Predict what the content-type of the response will be:

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

3) Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why?
   I saw that that piece of code was wrapped around h1 and h2 tags, which led me to believe that was the body. In addition, that piece of code was included in the code block of the get function, which the server will get when made a request.
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?
   The body is made up of strings, which led me to believe it was html text.

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:
   We will get the content of the array listed above (entries), which includes objects with id, date, and journal entry content.
2) Predict what the content-type of the response will be:
   Objects

- 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. I made that prediction because the variable "entries" was the argument of the send function. The entries array is listed above.
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?
   I was incorrect. It is an application/json. Content that is an array or object is sent as a json(javascript object notation).

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)
   This function is making journal entry objects for the user to post. It then pushes the object into the entries array. It then assigns an id number to the entry, which is + 1 from the previous entry. It then sends the entry to the user.

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)?
   Id, date, and journal entry content. They will be Json.
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" : "May 17", "content" : " :(" }
4) What URL will you be making this request to?
   http://localhost:4500/entry
5) Predict what you'll see as the body of the response:
   { "id": 3, "date" : "May 17", "content" : " :(" }
6) Predict what the content-type of the response will be:

- 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
      Json

7) Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why?
   No. I thought it would only show you the entry you made, but it shows you all the entries. Makes sense, since the code says send(entries)
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. because we are posting an object.

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)