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) **The Internet is a worldwide network of networks that uses the Internet protocol suite.**
2) What is the world wide web? (hint: here) **Interconnected system of public web pages 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? **Computers communicate and link together through networks.**
    b) What are servers? **Computers that store web pages, sites, and apps.**
    c) What are routers? **A signaler that makes sure information is passed correctly from one place to another.**
    d) What are packets? **Packets are parts of information that are sent and then reassembled in order to put everything together when received by the user.**
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 and web are like the human body, with the internet being the skeleton, and the rest being the web. The internet provides the foundation for everything else to be connected 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)

    https://www.figma.com/file/RMITDfHk9Vq57Urn94O7BC/HOW-THE-INTERNET-WORKS?node-id=0%3A1

Topic 2: IP Addresses and Domains

1) What is the difference between an IP address and a domain name? **An ip address is a numeric address that is assigned to you by your internet provider based on where you are, the domain name is a nickname for the space that you are visiting.**
2) What's devmountain.com's IP address? (Hint: use 'ping' in the terminal) **104.22.12.35**
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? **To keep your website secure it's important to not let them have access to everything just by knowing the IP address.**
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 checks the cache**

to see if the information is stored there, and if not then it sends out a query request until it can find the domain name and connect it to the IP address.

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)	The user has to click on the URL to start the process of retrieving information for the user.
HTML processing finishes	Request reaches app server	The request for information is sent out and reaches the server where the information is stored
App code finishes execution	App code finishes execution	The app finishes processing the request so that it can begin sending the return.
Initial request (link clicked, URL visited)	Browser receives HTML, begins processing	Next the browser can receive the information from the server, so that things can begin processing for the user.
Page rendered in browser	HTML processing finishes	This is what happens as everything is being put together for the user on the last end before the user sees the finished result of the URL displayed in entirety on it's page.
Browser receives HTML, begins processing	Pages rendered in browser	After the requests and responses have been sent out, the page can be viewed.

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: **Jurnni Journaling your journies**
2) Predict what the content-type of the response will be: **It will return back the header and the 200 to let the user know it's working.**

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

3) Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why? **Yes, because it was where the get was with the headers for HTML content**
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 because the first element was a string.**

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: **An array of objects, with ids, dates, and content.**
2) Predict what the content-type of the response will be: **January 2, Hello World;**
3) **January 2, Two days in a Row; June 12, whoops;**

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

4) 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 right about our prediction because we could see that entries was defined as an array of objects with the text inside.**
5) 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 sending back objects and we predicted that it would be in 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)
   -It defines a variable new entry.
   -It fills the variable new entry with an object containing the same keys as the entries variable.
   -It pushes the new variable entries into the new array.
   -double quotes and separated by commas. **It pushes the new variable entries into the new array.**
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 the date and content ones, they will be strings, in the code it says they are required.**
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}”: today, and “{body}”: name ‘**
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: **entries variable + new JSON**

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

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

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)