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)

   Network that connects computers all around the world

2) What is the world wide web? (hint: here)

   connected system of 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?

      i) Networks are a group of multiple computers that are interconnected that allows the computers to share resources and enables them to communicate.

   b) What are servers?

      Computer that hosts information or files that another computer is requesting

   c) What are routers?

      i) Routers are devices that connect several computers to the network

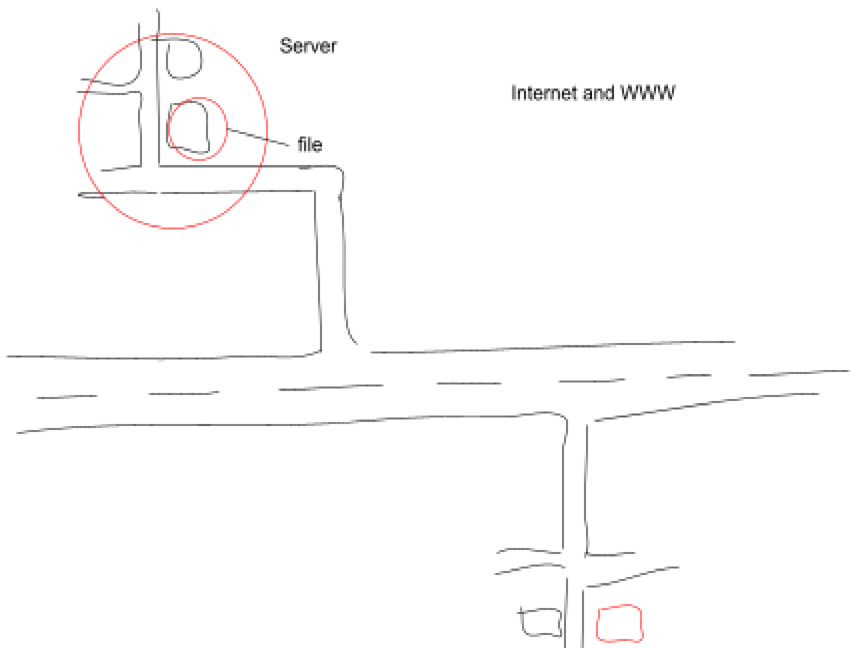
   d) What are packets?

      Units of data that are used to transfer files and info from one computer to another

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 network of roads and highways. The roads are like servers/computers and the highway can be viewed as a network that connects those servers together. Individual stores/houses are like files, you can use the roads and highways to share resources from one location to another.

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?

   The IP address is the location of the resource that allows computer to identify each other. The domain name is a name for that address that is linked to a website
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 protect private data and not allow anyone to make unauthorized changes to the data on the site.
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)

   Client computer requests the IP address associated with a specific domain name from a DNS, and the DNS returns the IP address for that website

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)	a request has to be initiated before anything will happen
HTML processing finishes	Request reaches app server	server won't send anything until it receives the request
App code finishes execution	App code finishes execution	I put this step before the rest because the app has to finish executing the code before we can process the HTML
Initial request (link clicked, URL visited)	Browser receives HTML, begins processing	We put this fourth because the HTML has to be received before we can process it
Page rendered in browser	HTML processing finishes	The HTML processes after it is received
Browser receives HTML, begins processing	Page rendered in browser	The page is rendered once the HTML is finished processing

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

- 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, we can see in the code that app.get for the '/' will print the body text
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? We were incorrect, we just guessed that it would be JSON. It was text/html

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: Whatever argument is passed to the 'entries' parameter
2) Predict what the content-type of the response will be: JSON

- 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. The command returned the journal entries.
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 were correct about the content type. We predicted that it would give us data in the form of JSON since its an object

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)
   1. Creates a new journal entry
   2. Pushes that new entry to the entries object
   3. Posts that updated entry object on the server
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: number
   date: string
   content: string

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.

    Let newEntry = {
    "id" : 23456,
    "date" : "'June 19",
    "content" : "Dylan and Zenith lab work"
    }

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":23456,"date":"June 19","content":"Dylan and Zenith lab work"}
6) Predict what the content-type of the response will be: 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  -X POST -H 'Content-type: application/json' -d '{"id" : 23, "date" : "June19", "content":"DylanandZenith"}'
      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?

    We were mostly correct about the body, but we didn't expect the id number to increase on its own

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, the content type was JSON, we specified that it would be JSON when we made the request.

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)