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.

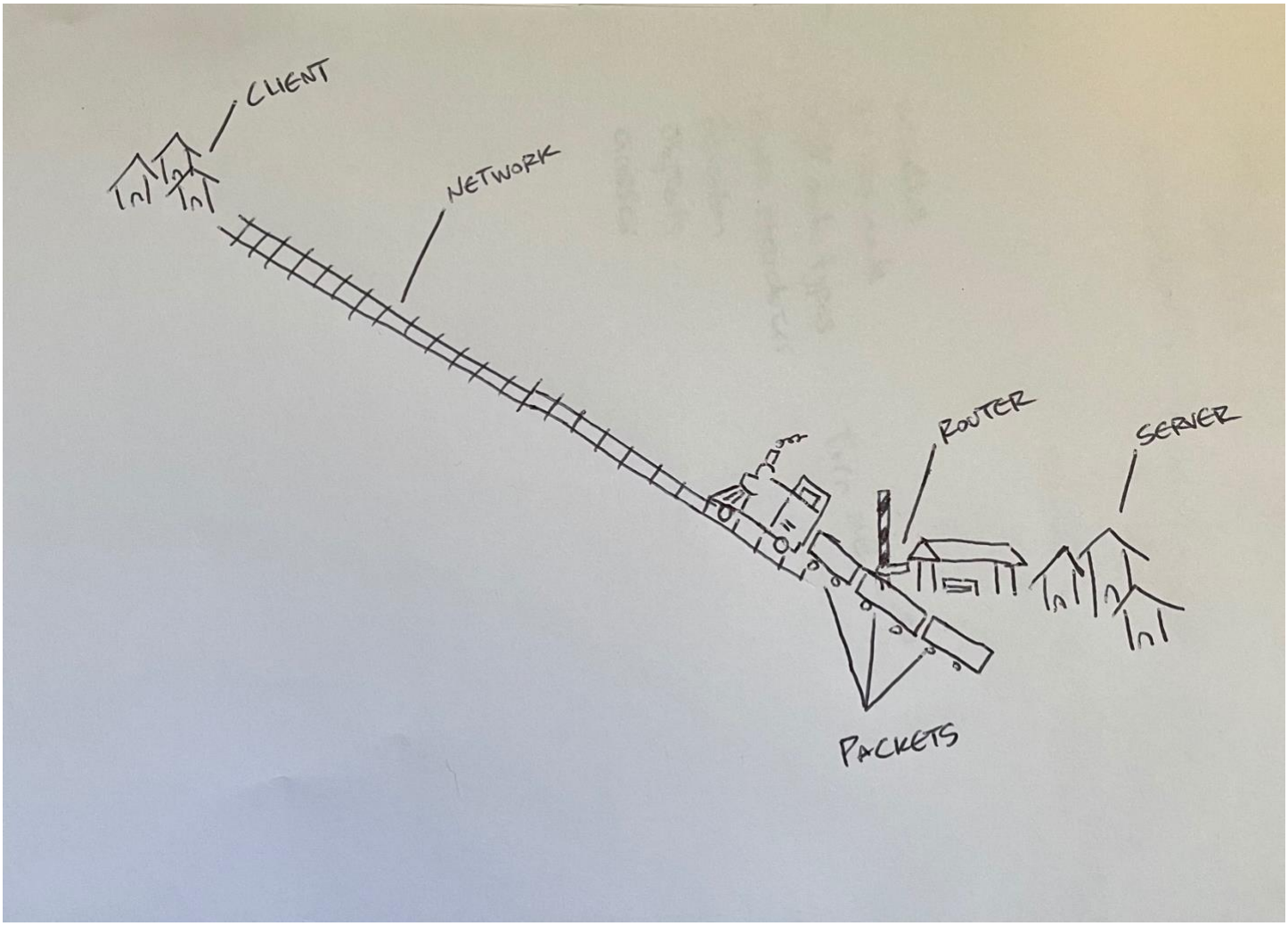
2) What is the world wide web? (hint: here) is an interconnected system of public webpages accessible through the Internet.

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? Communication between a number of computers.
    b) What are servers? Computers that store webpages, sites, and apps for clients to access
    c) What are routers? A special computer that connects to a network of computers, sending messages between each computer wirelessly. It can also connect to other routers to communicate with other networks.
    d) What are packets? Small chunks of data that are sent from the server to the client.

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 metaphor that we came up with is a train route between 2 villages. The network is the train tracks connected between the villages, allowing communication and movement. One village would be the client, and the other village would be the server that the passengers want to reach. The router is the signaler at a railway station, sending the trains to the right destination. And the packets is the train itself which is made up of different cars, sending the passengers from one train station to another. If this is so, then the internet would be a series of train routes encompassing not only the two villages mentioned but also other villages via different train stops and railways.

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 numerical address assigned to a website; whereas the domain name is a part of the URL, which points to the IP address.
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? For data security purposes.
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) Using DNS, the website uses servers that match up a web address (ex: google.com) to the IP address so when we go to the web address it goes 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)	I put this step down first because the user must request something for any action to happen.
HTML processing finishes	Request reaches app server	I selected this as the second step because as the web address is sent to the DNS to find the IP address
App code finishes execution	App code finishes execution	The next step is the DNS returns with the IP address requested
Initial request (link clicked, URL visited)	Browser receives HTML, begins processing	I placed this as the 4th step because as the IP address loads, in the browser, it loads the content of the website.
Page rendered in browser	HTML processing finishes	I placed this as the step before last because now the packets of data processing has finished.
Browser receives HTML, begins processing	Page rendered in browser	This is the last step because the website has been fully loaded in the browser as the outcome of the request.

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: I predict that the body will say "Jurrni – journaling your journies"
2) Predict what the content-type of the response will be: I predict that the content type will be plain html text.

- 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, I was correct. I referenced the code to make my prediction.
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, I was correct. I referenced the code to make my prediction.

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: The body of the response will be entries of data regarding ID, DATE, and content.
2) Predict what the content-type of the response will be: plain html text

- 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 was correct. I referenced the code (app.post('/entries') to make my prediction.
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, I was correct. I referenced the code to make my prediction.

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)
   The function is doing several different commands: get, post, and listen.
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 CONTENT strings
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.

```
{
    "id": 0,

  },
```

4) What URL will you be making this request to? I will be posting to http://localhost:4500
5) Predict what you'll see as the body of the response: I think I will see a body of the response that contains thee journal entries.
6) Predict what the content-type of the response will be: plain html text
- 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
5) Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why? I predict that I will be able to see the entries of data regarding ID, DATE, and content, with an addition of the object added in step 5.
6) 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. I was able to predict that because of the data in the code provided as well as the information I added.

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)