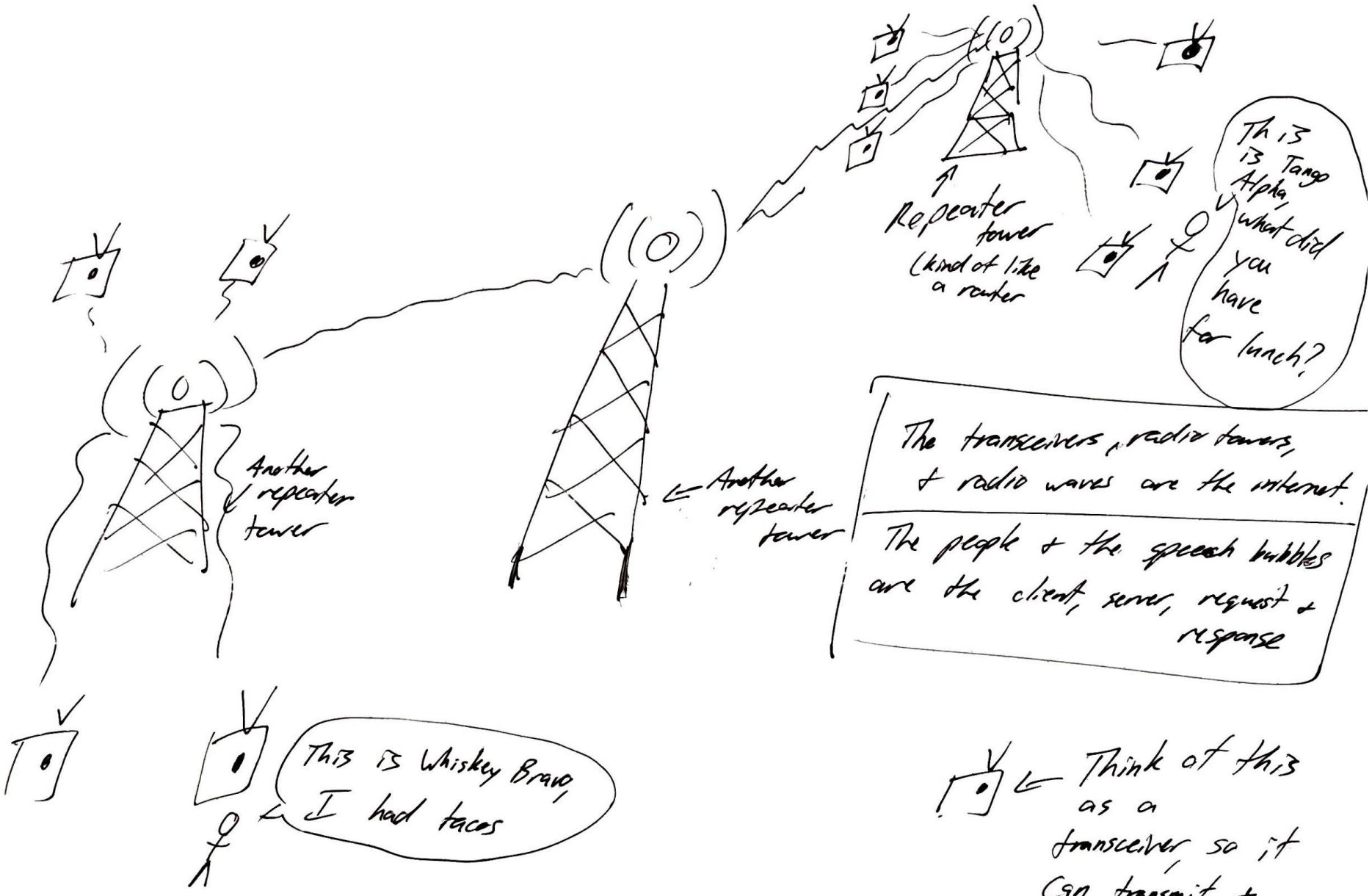
How the Web Works

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)
   ANSWER: The Internet is a network of networks; it is the technical connections that bind computers and computer networks together.
2) What is the world wide web? (hint: here)
   ANSWER: The World-Wide Web are the sites, pages, and information that are accessible through the Internet (see above)
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?
      ANSWER: They are a group of computers that are linked/connected, either physically or wirelessly
   b) What are servers?
      ANSWER: computers that store web pages, sites, or apps
   c) What are routers?
      ANSWER: smaller computers that function as a central hub of a network, linking computers together.
   d) What are packets?
      ANSWER: A tool to send data in small chunks across the web.
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)
   ANSWER: A radio network can represent the internet, with transceivers as the client computers and as servers. Repeater towers can act as routers and modems. A person on one transceiver can request information from another radio operator with the signal being broadcast through the repeater towers. A radio operator's callsign is like their IP address.
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)


Repeater tower (kind of like a router
This is Tango Alpha, what did you have for lunch?
Another repeater tower
Another repeater tower
The transceivers, radio towers, & radio waves are the internet.
The people & the speech bubbles are the client, server, request & response
This is Whiskey Bravo, I had tacos
Think of this as a transceiver so it can transmit & receive messages

Topic 2: IP Addresses and Domains

1) What is the difference between an IP address and a domain name?
   ANSWER: The IP address is the actual address (which is a set of 4 numbers separated by dots), and the domain name is a nickname for the IP address that a human can easily remember.
2) What's devmountain.com's IP address? (Hint: use 'ping' in the terminal)
   ANSWER: 104.22.12.35 but then I tried it again later and got this: 172.67.9.59
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?
   ANSWER: If they access your site by directly typing in the IP address, they might not reach the content they are trying to find (if you don't have anything on domainname.com, but you do have something for domainname.com/home) That's just a guess. When I asked the Internet, people mentioned it was more secure, but failed to give details as to why that was. The ever-so-wise Internet also said that it could lead to the creation of duplicates or duplicate information.
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)
   ANSWER: The Web Browser first asks the local computer's DNS if it already knows the IP address. If not, then the browser asks the router's DNS

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 literally says, "Initial" in it, sooo… it is the first request
HTML processing finishes	Request reaches app server	Then the request, the one from step 1, reaches its destination
App code finishes execution	App code finishes execution	The APP server then figures out what the request wants and packages it all nice and neat, and sends it back to the client.
Initial request (link clicked, URL visited)	Browser receives HTML, begins processing	The client receives the requested data from the server and starts unpacking it.
Page rendered in browser	HTML processing finishes	It finishes unpacking everything
Browser receives HTML, begins processing	Page rendered in browser	Finished product is displayed on the screen.

I actually did this last. The exercise below helped me figure it out.

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
  Predict what you'll see as the body of the response: The version of the HTTP used, the response code, any headers, the content type, and the date/time the server is at.
  Predict what the content-type of the response will be: text/html
- Open a terminal window and run `curl -i http://localhost:4500`
  Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why? No, I misunderstood what the body meant. I got back the html text
  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 because I saw in the code that it was meant to produce an html response.

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: This time we'll get an array of the objects that are in the js code.
2) Predict what the content-type of the response will be: application/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, I was. I guessed that it would be so because I could see in the code that it would send the entries variable, which is an array of objects.
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. Same explanation as I gave above.

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)
   ANSWER: This function creates a new object, adds that object to the entries array, increases globalId by one, then sends the updated entries array back to us.
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)?
   ANSWER: the body object should include a content property and a date property, and they should be 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. '{ "content" : "Boo-yah!!", "date" : "August 2"}'
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: an updated entries array, with the new object tacked on the end.
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? Yes, yes I was. I made that prediction because the function sends the entries array, just like the get function.
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, yes I was. It sent the same variable as the get function, so it would be the same content type.

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)