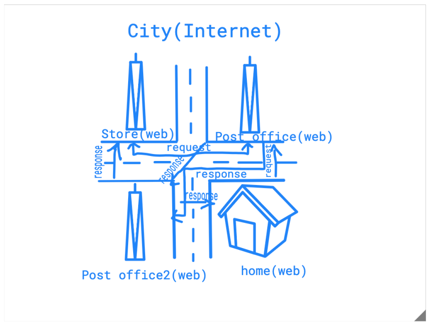
How the Web Works

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) A: A network that connects computers together worldwide
2) What is the world wide web? (hint: here) A: connected system of webpages 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: when two or more computers communicate**
    b) What are servers? **A: Computers that store webpages, sites, or apps**
    c) What are routers? **A: A special tiny computer that's connected to your main computer, a translator**
    d) What are packets? **A: Smaller parts of a larger message sent over computer networks**
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: **Think of the Internet as a city, and the web are all the stores, buildings, road's, everything that makes the city what it is, and all of the stores and roads are relying on the city.**
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? **A: an ip address is directly tied to your computer, and a domain name functions as a link to your ip address, or vise versa**
2) What's devmountain.com's IP address? (Hint: use 'ping' in the terminal) **A: (172.66.40.149)**
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? **A: because Dev Mountain is part of a network that shares the same ip address as other websites on a shared network**
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) **A: because your computers ip address is directly linked to the website ip address you search, but you just search it with**

words instead of the ip address, because that's how the two are able to connect. If the ip address is stored in your cache memory

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)	Keyword: initial request. You need to make a request in order for anything to happen.
HTML processing finishes	Request reaches app server	The server needs the request in order to know what to execute.
App code finishes execution	App code finishes execution	I put this after "request reaches app server" because you need a request to come in to know what to execute.
Initial request (link clicked, URL visited)	Browser receives HTML, begins processing	I put this before "HTML processing finishes" because this is the initial process to render the page before it's displayed.
Page rendered in browser	HTML processing finishes	I put this before "page rendered in browser" because you need to finish processing all the html before you display it.
Browser receives HTML, begins processing	Page rendered in browser	I put this last because it's the end step, the end goal.

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**
2) Predict what the content-type of the response will be: **Journaling your journies**

- 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? **A: Yes. Because <h1> and <h2> typically get put in the body in html.**
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? **A: No, because I thought the content type was just part of the body, instead of it being the type of data/language displayed.**

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: **A: The entries array**
2) Predict what the content-type of the response will be: **A: HTML**

- 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? **A: yes, because /entries was being requested, and it matched up with the entries array**
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? **A: No, I thought it was going to be in HTML like the last request, not 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) **A: Allowing user to add a new entry, and allowing it to be pushed into the entries array, and creating an id for it, and finally, sending the request to the entries 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)? **A: you need an id, date, and content.**
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.
4) What URL will you be making this request to? **A: http:/localhost:4500/entry**
5) Predict what you'll see as the body of the response: **A: the entries array, with the "new entry" in the array**
6) Predict what the content-type of the response will be: **A: 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? **A: Yes, I used a previous JSON request we did in in the interactive lecture, and replaced the url with the current entry url, and changed the information to align with the entry array.**
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? **A: Yes, it was JSON because we made our request in JSON, so it would've had to have been the same in order to display it.**

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)