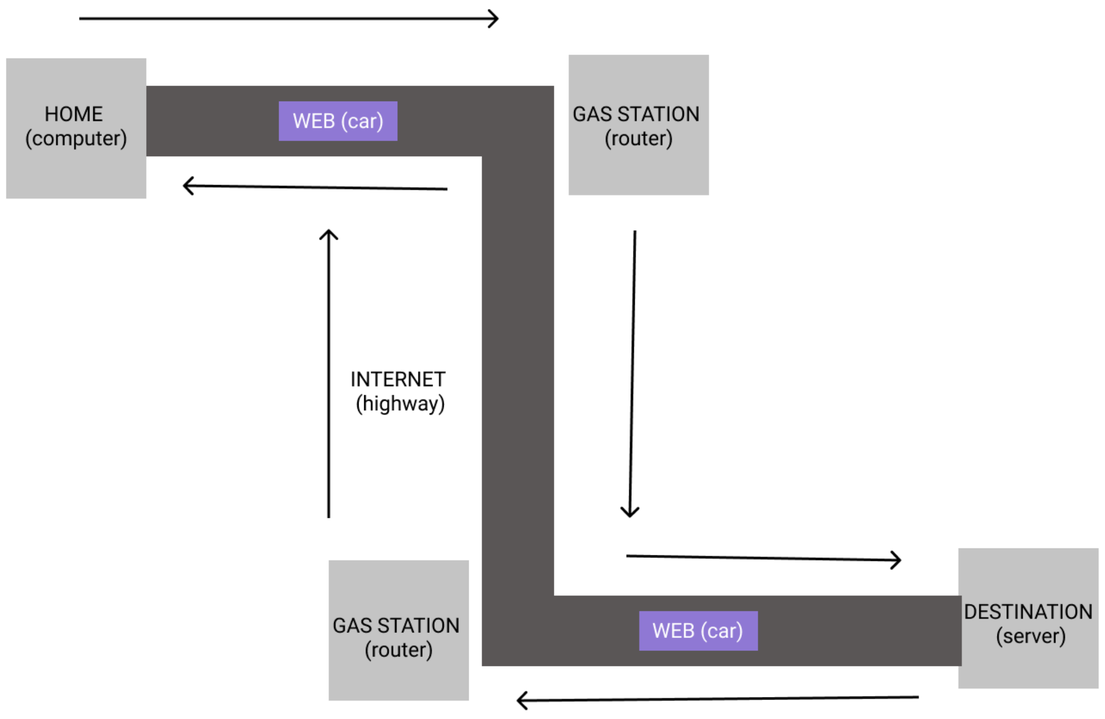
How the Web Works

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
2) **What is the world wide web? (hint: here)** an application 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?** Routers connecting to routers
    b) **What are servers?** Computers that keep and store information (ex: webpages, apps, and sites)
    c) **What are routers?** Mini computer that makes sure that a message sent from a given computer arrives at the right destination computer
    d) **What are packets?** Formatted chunks of data sent from the server to the requested client; usually in smaller chunks rather than one big set
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 highway that gets you from one location to your destination; and the web is the vehicles transporting passengers/items from one place 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 a set of numerical instructions and the domain name is a link to the IP address
2) **What's devmountain.com's IP address? (Hint: use 'ping' in the terminal)** 104.22.13.35
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?** Security purposes: protection of information from users as well as tracking user 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)**
The browsers keep the IP addresses cached within the database for quicker recall, if the IP is not within the browser, it will go though the server to retrieve it

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 first because it is initiating the request
HTML processing finishes	Request reaches app server	I put this step after the initial request because the request has to reach the server before bringing back the info
App code finishes execution	App code finishes execution	I put this step before HTML because the app code produces the requested app prior to the code being processed
~~Initial request (link clicked, URL visited)~~	Browser receives HTML, begins processing	I put this step next because it is the initiation of the code of the program after the app has been received locally
~~Page rendered in browser~~	HTML processing finishes	I put this step next because it is the final processing of the code of the program prior to the page interaction
Browser receives HTML, begins processing	Page rendered in browser	I put this step last because it is the final product of the request, the web page that was requested

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:** details about the web page
2) **Predict what the content-type of the response will be:** HTML format

- **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?** Based on the google example I thought the content would unreadable but this web page is much less detailed and therefore more readable in terminal
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, most web pages are formatted in HTML so I assumed a simple web page like this would be as well.

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:** we will see the currently entered entries form the above code
2) **Predict what the content-type of the response will be:** text formatted like an object

- **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, based on the function in the code, it output the entries from above
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, the way the code was formatted, it looked like an object format for the function of '/entry', so we expected terminal to output it as 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)** creating a new entry object
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:
        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?**
5) **Predict what you'll see as the body of the response:** id: 3, date: September 13, "entered entry"
6) **Predict what the content-type of the response will be:** it will return a single object with our updated info

- **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**
    - curl -i -X POST -H 'Content-type'{"id":3, "date":"September 13", "content":"yay web dev"}' 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?** Kind of, it returned all of the previous entries in addition to the newly added one as well
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?** For the most part, I forgot that all of the json info would also be included, but that my new entry would also be included.

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)