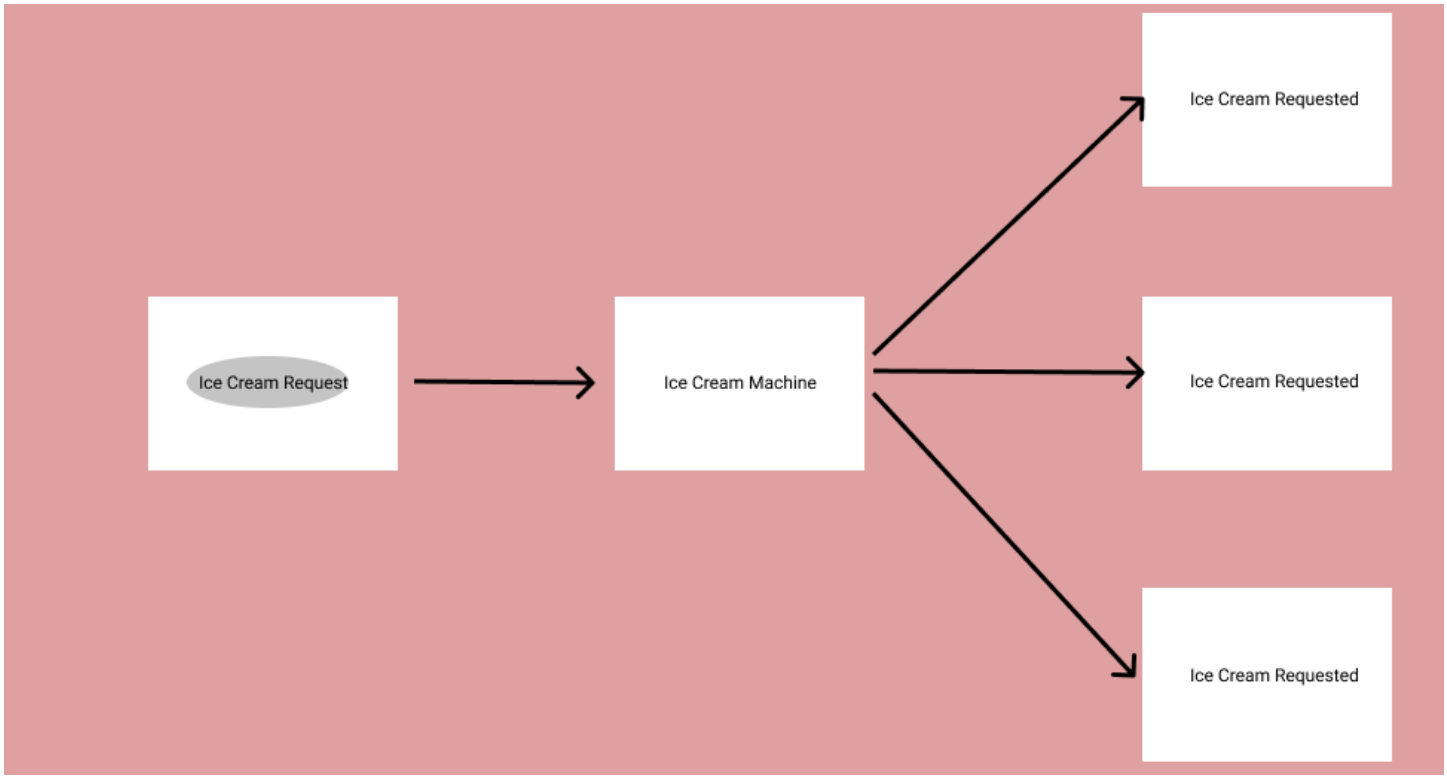
# How the Web Works

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 very large network of networks.
2) What is the world wide web? (hint: here)
    a) A series of public 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? A system of connected computers, using ethernet, wifi or bluetooth.
    b) What are servers? Servers are computers that store web pages, sites or apps.
    c) What are routers? A computer whose one job is to direct messages sent between other computers.
    d) What are packets? A series of small data chunks that are sent from a server to a 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)
    a) The internet is an ice cream machine and the web is ice cream.
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) **IP address is the identifier for the computer's location while the Domain name is the human readable alias.**
2) What's devmountain.com's IP address? (Hint: use 'ping' in the terminal)
    a) **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?
    a) **Prevents overloading hacking attempts to overload the website by spamming its IP**
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) **The browser uses DNS to pair an IP with a Domain Name. It will check through cached data first before reaching out to the resolver.**

## 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 ____*<br>*- I put this step before/after ____ because ____* |
| Request reaches app server | Initial request (link clicked, URL visited) | Has to receive a request to start the process of loading a webpage. |
| HTML processing finishes | Request reaches app server | App server needs to receive requests before we have code to process. |
| App code finishes execution | Browser receives HTML, begins processing | Once the app server has responded to the request we will have code to start processing. |
| Initial request (link clicked, URL visited) | HTML processing finishes | Nothing can be done until the code has finished processing. |
| Page rendered in browser | Page rendered in browser | Has to have the information loaded before anything can be rendered. |
| Browser receives HTML, begins processing | App code finishes execution | This marks the end 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:
    a) **Entries will be the body.**
2) Predict what the content-type of the response will be:
    a) **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?
    a) **No the body is 'journaling your journeys'**
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 we were not, we just kinda shot in the dark. The content type was HTML/Text.**

*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) **We will see the entries.**
2) Predict what the content-type of the response will be:
    a) **text/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? **We were correct about the body. We predicted this based on how we were calling for the entries content specifically.**
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? **No the content type was application/JSON. We predicted purely based on the results of the last curl.**

*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) **Defining a new object (newEntry)**
    b) **Pushes the object into the array**
    c) **Giving the object a different id number in ascending numerical order.**
    d) **Reports the status**
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) **The objects needs a global id, a date and a string for content**
    b) **Global ID: number, Date: string and number, 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.
    a) **ID: "3", Date: "July" 7, String: "My Birthday"**
4) What URL will you be making this request to?
    a) **http://localhost:4500/entries**
5) Predict what you'll see as the body of the response: **The entries**
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? **Since we were adding the entries we figured that would be reported back.**
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 we were right. We had previously seen that the body was application/JSON and felt it would maintain that style.**

## 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)