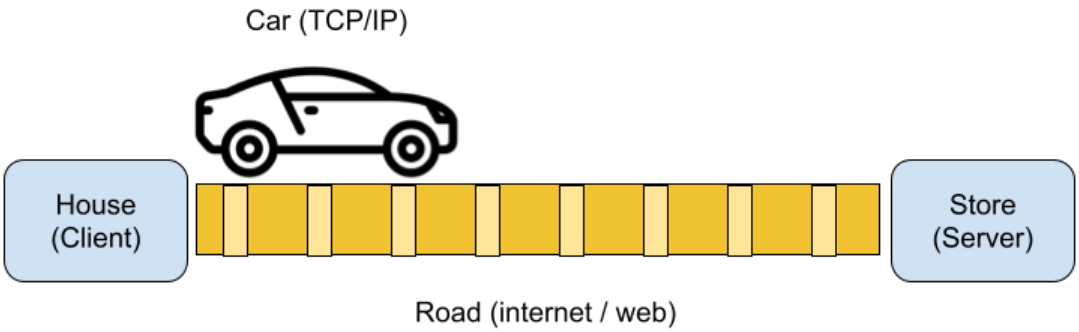
# How the Web Works

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 of computers.
2) What is the world wide web? (hint: here)
    a) 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?
        i) Networks are computers connected by a router.
    b) What are servers?
        i) Computers that store component files of webpages – code files and assets.
    c) What are routers?
        i) A computer that ensures a message gets from one computer to another.
    d) What are packets?
        i) Small chunks of data 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)
    a) The internet can be compared to a road or highway that connects a person's house to a store. The store contains a product, which are like component files (code files and assets). TCP/IP is a car that transports the products from the store to the person's house. DNS can be compared to the (archaic) yellow pages that contain store addresses. Packets can be the shopping bags containing the products.
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) The IP address is the number assigned to the server, while the domain is the website's name address, which is more accessible and easier to remember.
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) Might be easier to hack with a direct IP address.
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 checks its cache (memory) to see if it knows the IP address. If it does not, it checks the Local Router DNA. If that fails, it checks the ISP DNS, and if that fails it checks the Root DNS servers.

## 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) | The URL is the first step on the client's side. |
| HTML processing finishes | Request reaches app server | Request queuing on the server |
| App code finishes execution | App code finishes execution | Once the server receives the request, it executes the app code on its computers. |
| Initial request (link clicked, URL visited) | Browser receives HTML, begins processing | Data packet containing code data arrives on the client side and the browser processes it. |
| Page rendered in browser | HTML processing finishes | Component files are parsed by the browser. |
| Browser receives HTML, begins processing | Page rendered in browser | The client sees the final results – the webpage. |

## 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) We'll probably see an H1 heading that says "Jurrni" and an H2 heading that says "Journaling your journies".
2) Predict what the content-type of the response will be:
    a) Probably text/html.

- 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, we were correct. We made that prediction because we were looking for the GET method line of code, with the '/' (root) directory.
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) Yes. Once we found the correct line of code, the .send method contained h1 and h2 tags, which are HTML tags.

*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 predict that we'll see the objects within the 'entries' array as a text which includes the { }.
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?
    a) We were off. The body actually contains JSON data and it's displayed together with the square brackets [] outside the curly brackets {}. We are not proficient yet at reading back-end server-side code, which apparently includes JSON.
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) The content type ended up being "application/json". Apparently it's using line #4 which call upon the express.json method. We're not yet familiar with this method.

*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) The "entries.push" method adds the object contained within the "newEntry" varible onto the "entries" array.

b) “globalID++” simply adds one more object to the counter variable “globalID” to denote that a new (fourth, in this case) object has been added to the array.
c) res.status(200) is the HTTP response code indicating the request has been successful.
d) res.send(entries) displays the “entries” array onto the client-side browser.

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) [Did not cover back-end code yet]
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) Localhost or 127.0.0.1
5) Predict what you’ll see as the body of the response:
6) Predict what the content-type of the response will be:

- 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: application/json’ -d newEntry  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?
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?

## 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)