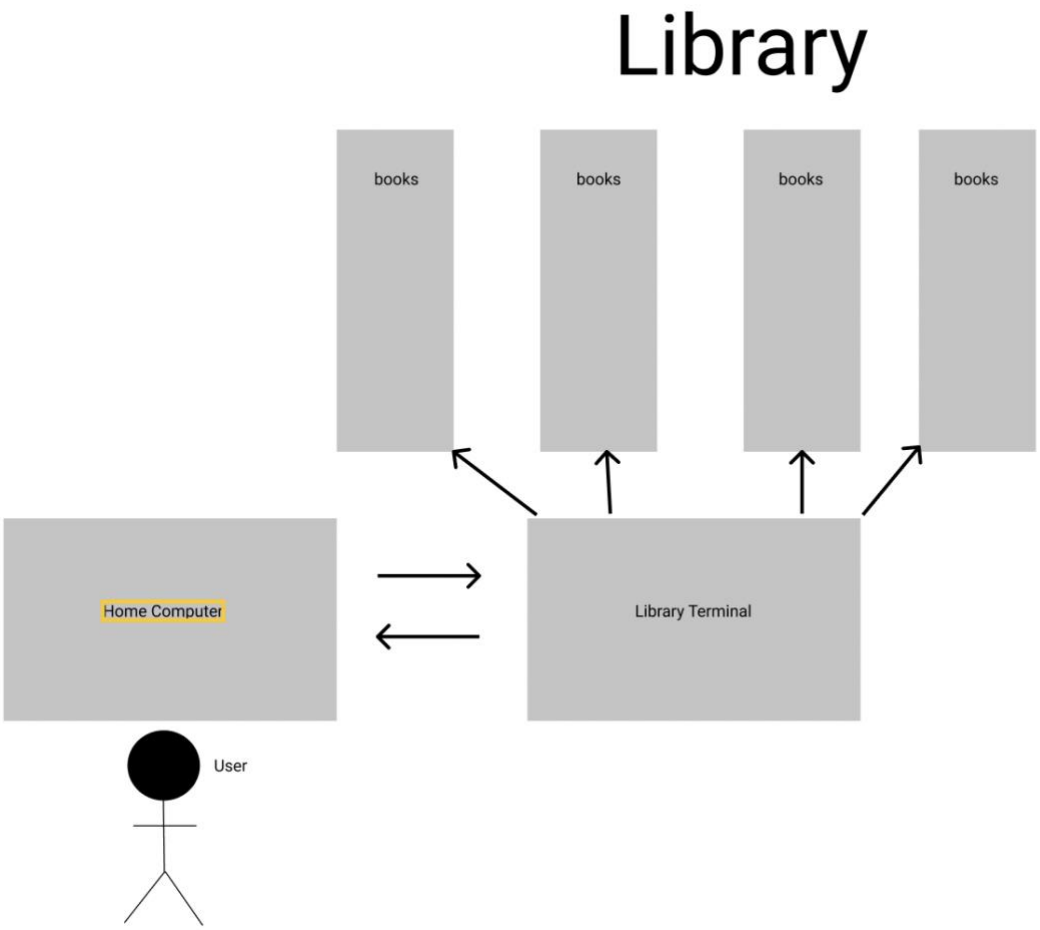
How the Web Works

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 that uses the Internet protocol suite (also named TCP/IP from its two most important protocols.
2) What is the world wide web? (hint: here) Referred to as WWW, W3, or the Web is an interconnected system of public webpages accessible through the Internet. The web is not the same as the internet: web is one of many applications built on top of 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 collection of computers, servers, mainframes, network devices to allow data sharing.
    b) What are servers? Servers are computers that store webpages, site, or apps. When a client device wants to access a webpage, a copy of the webpage is downloaded from the server onto the client machine to be displayed in the user's web browser.
    c) What are routers? Receives and sends data on computer networks. Device that communicates between the internet and the devices in your home that connect to the internet. Direct traffic.
    d) What are packets? Packets are a small segment of a larger message. Data sent off computer networks, such as the internet.
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) – Water System, Library, Road System
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 difference between an IP address and a domain name is that an IP address has a certain line of numbers that are hard to memorize, domain names were created for this reason to make it easier to navigate to a website.
2) What's devmountain.com's IP address? (Hint: use 'ping' in the terminal) 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? The main reason is for security of the websites. So that users can't edit it or change it.
4) How do our browsers know the IP address of a website when we type in its domain name? Browsers use a dns service which stores the ip addresses to the browser cache and the domain names that they are linked to. Browsers may also cache this for faster retrieval. (If you need a refresher, go read this comic linked in the handout from this lecture)

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 in order to load a webpage, you will need to type out a website link and request access once you click enter.
HTML processing finishes	Request reaches app server	The request would be routed to the location requested.
App code finishes execution	App code finishes execution	This step processes data and is sent to the client.
Initial request (link clicked, URL visited)	Browser receives HTML, begins processing	This step is next since the request has finally reached the computer server.
Page rendered in browser	HTML processing finishes	The HTML processing is finally finished once it fully reaches the server.
Browser receives HTML, begins processing	Page rendered in browser	This is the last step because the page is finally rendered, after HTML CSS and javascript.

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: The misspelling of Jurrni, "journaling your journies"
2) Predict what the content-type of the response will be: It is a string involving back ticks.

- 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? Yes, it was because of the get function and the res.status input in the line of code. Also, the /entries.
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? Sort of? We knew it was strings, but the input was text/html.

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: It would give the objects of the code, such as date, content and id.
2) Predict what the content-type of the response will be: Content type will be 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? Yes, because of the given and the entire at the top as objects. Entries was used and seeing the let function.
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, but seeing it now it's understandable that it's application/json because of the entries that were typed in the json file.

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)

- This function is creating a new entry and pushing it into 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)? Body and date will need to be included.
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? http://localhost:4500/entry
5) Predict what you'll see as the body of the response: The response will be what we inputted for the double quotes and separated by commas. Journal entries, dates, etc.
6) Predict what the content-type of the response will be: The content-type would 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
    - curl -i -X POST -H 'Content-type: application/json' -d '{"date": "June 13", "content": "Back at it"}' 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? Yes, it was due to the fact of the object arrays and that we'll be adding new stuff to the body.
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, but interesting to see what we needed to add to the json file.

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)