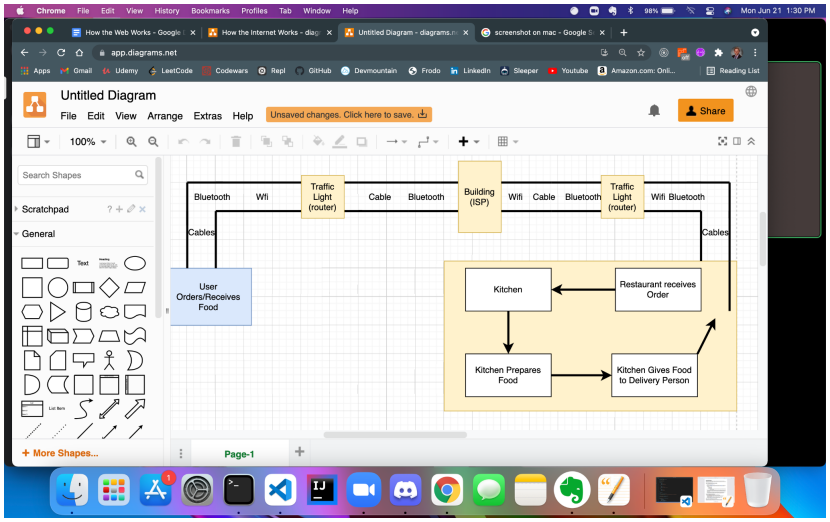
How the Web Works

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

- What is the internet? (hint: here)
    - The internet is a worldwide network of networks that uses the Internet protocol suite.
- What is the world wide web? (hint: here)
    - An interconnected system of public web pages accessible through the Internet. The Web is not the same as the Internet: the Web is one of many applications built on top of the Internet.
- 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
    - What are networks?
        - A network is how computers connect with one another through cables, routers, telephone lines, internet, or bluetooth. The Internet is a technical infrastructure which allows billions of computers to be connected all together.
    - What are servers?
        - Computers that store web pages, sites, or apps. It is responsible for sending the necessary files for that particular website to the user.
    - What are routers?
        - Each computer on a network is connected to a special tiny computer called a *router*. This *router* has only one job: like a signaler at a railway station, it makes sure that a message sent from a given computer arrives at the right destination computer.
    - What are packets?
        - Format in which the data is sent from server to client. It is sent in small chunks to make the web more efficient.
- 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/network is comparable to road and building structures. This structure helps people (or computers) receive anything from a remote location, such as ordering food from a restaurant. Once a user orders food from a restaurant (like sending a request to a server) the order goes through the user's network(cables, routers, isp). The kitchen (the server) receives that order and serves the food which is then delivered back to the user through the network (requested web pages).
- 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

- What is the difference between an IP address and a domain name?
    - The domain name is the shorthand(nickname) of the IP address. The IP address is the actual location of the server the user is trying to access.
- What's devmountain.com's IP address? (Hint: use 'ping' in the terminal)
    - 172.67.9.59 (Utah County)
    - 104.22.12.35 (Dallas)
- 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?
    - If, for some reason, the IP address should have to change, you have no way of redirecting users. Whereas with a domain name, it's simply a matter of updating DNS records to point to the new IP address.
- 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)
    - Through the DNS Server.

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.

Example: Here is an example step	*Here is an example step*	*- I put this step first because ____* *- I put this step before/after ____ because ____*
Initial request (link clicked, URL visited)	Link clicked/Domain requested to view	The user must make a request before any files can be sent.
Request reaches app server	Requested	Server has to send back the files.
Browser receives HTML, begins processing	Received	Once the user receives the files, the browser starts processing the return information.

HTML processing finishes	Processed	Once the HTML has finished processing, it will start to render the CSS and JavaScript.
App code finishes execution	Finished	The CSS and JavaScript finishes processing.
Page rendered in browser	Rendered	Once all of the files have been processed, the page will render.

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
- Predict what you'll see as the body of the response:
    - <h1>Jurrni</h1><h2>Journaling your journeys</h2>
- Predict what the content-type of the response will be:
    - HTML with a 200 status code.
- Open a terminal window and run `curl -i http:localhost:4500`
- Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why?
    - Yes, we looked at the GET route in server.js.
- 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 content was wrapped in h1 and h2 tags located in the GEt route in server.js.

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.
- Predict what you'll see as the body of the response: entry's array
- Predict what the content-type of the response will be: 'A String'
- In your terminal, run a curl command to get request this server for /entries
- 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 pointing to the entries array inside server.js.

- 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, it was a string in JSON format.

Part C: POST /entry

- Last, read over the function that runs a post request.
- At a base level, what is this function doing? (There are four parts to this)
    - This function is letting a user post an entry (id, date, content). This entry is then pushed to the newEntry array. After the user posts their next id will go up by one. The updated entries array will be sent to the user.
- 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 (number), date (string), content (string).
- 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.
    - '{"id": "3", "date": "June 21", "content": "Hello there!"}'
- What URL will you be making this request to?
    - http://localhost4500/entry
- Predict what you'll see as the body of the response:
    - The entries array.
- Predict what the content-type of the response will be:
    - 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
- Were you correct about the body? If yes, how/why did you make your prediction? If not, what was it and why?
    - Yes, it returned the new information we posted to the entries array.
- 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, it was in JSON format.

Submission

- Save this document as a PDF
- Go to Github and create a new repository. (Click the little + in the upper right hand corner.)
- Name your repository "web-works" (or something like that).
- Click "uploading an existing file" under the "Quick setup heading".
- Choose your web works PDF document to upload.
- Add "commit message" under the heading "Commit changes". A good commit message would be something like "Adding web works problems."
- 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)