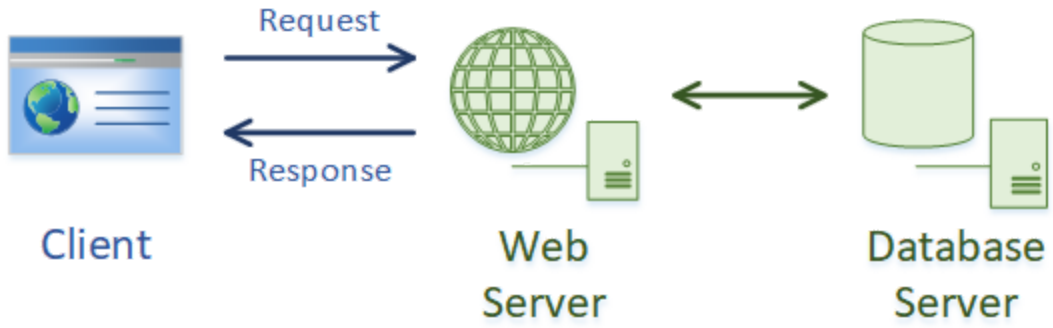
How the Web Works

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
2) What is the world wide web? (hint: here)
   The World Wide Web is an information system where documents and resources are interlinked and accessible across 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? Different levels of computers that can link and communicate
   b) What are servers? Servers can send messages intelligible to web browsers.
   c) What are routers? Routers make sure that a message sent from a given computer arrives at the right destination computer.
   d) What are packets? Small chunks of webfiles that are sent and requested when viewing the web
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 and web are the same as Forge mode in Halo. The web represents all the possibilities that forge offers, while the creation of something on a map is the internet within the framework of the web.
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 an actual set of numerical instructions, the domain name functions as a link to the IP address
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

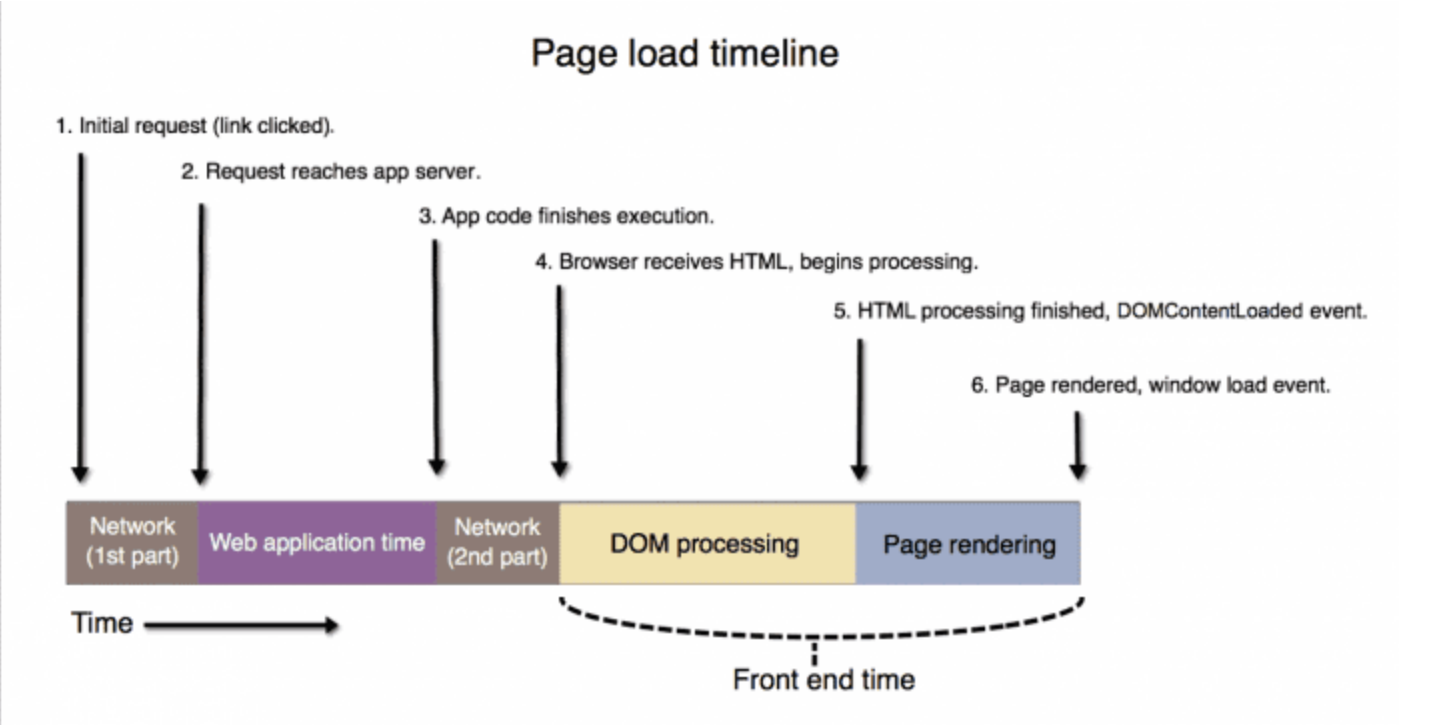
directly at the IP address? One reason is that you would need a static IP address in order to make sure the IP never changes. If it did change you would have no way of redirecting users. With a domain name you can just update the DNS records to point to the new IP

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)
   The Domain Name System (DNS) links the name with an IP address

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 ____*
HTML processing finishes	Initial request (link clicked, URL visited)	I put this step here because this is the initial step.
App code finishes execution	App code finishes execution	I put this step here because the information web page has to request and load its information included on the page.
Initial request (link clicked, URL visited)	Browser receives HTML, begins processing	I put this step here because once the information is received, the web page begins formatting it all.
Page rendered in browser	HTML processing finishes	I put this step here because it is the final formatting of information before the web page loads.
Browser receives HTML, begins processing	Page rendered in browser	I put this step here because it is the product of a completely loaded page.



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:
   Jurrni
   Journaling your journies
2) Predict what the content-type of the response will be:
   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?
   Yes. We looked at the code .get portion with the forward slash parameter. Looking within, we saw HTML tags present and deduced information within to be text.
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. Upon looking within the app.get element we saw HTML tags and the information within to be 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:
   We will see the object "entries"
2) Predict what the content-type of the response will be:
   We will see everything defined in the "let entries =" 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, looking at the .get(/entries) function we saw it pointed the website to the object "entries". We looked within the object "entries" and wrote that out.
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 code "let entries =" shows everything we will see when the entries page is opened.

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)
   The function creates a newEntry object
   The function adds 'newEntry' to the array
   The function increases the globalId by 1
   The function reads a status of 200 and pushes out the data from the object "entries"

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)?
   Properties will include: id, date, and content.
   The datatypes will be numbers and text.

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.
   curl -i -X POST -H 'Content-type: application/json' -d '{"id": "4", "date": "September 13", "content": "'We did it!!!"}' http://localhost:4500/entries
   curl -i -X POST -H 'Content-type: application/json' -d '{"name": "Large Jedi"}'
   http://localhost:4000/api/list

4) What URL will you be making this request to?
   http://localhost:4500/entries
5) Predict what you'll see as the body of the response: Everything in the entries array including our added entry
6) Predict what the content-type of the response will be:
   application/json; charset=utf-8

- 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? Yes, the page showed the previous entries along with our new object.
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 figured it would work like it did with the previous entries

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)