**Demo**

These links are good lecture prep:

<https://www.youtube.com/watch?v=3QhU9jd03a0&list=PL8dPuuaLjXtNlUrzyH5r6jN9ulIgZBpdo&index=29>

<https://www.youtube.com/watch?v=AEaKrq3SpW8&list=PL8dPuuaLjXtNlUrzyH5r6jN9ulIgZBpdo&index=30>

<https://www.youtube.com/watch?v=guvsH5OFizE&list=PL8dPuuaLjXtNlUrzyH5r6jN9ulIgZBpdo&index=31>

<https://developer.mozilla.org/en-US/docs/Learn/Common_questions/How_does_the_Internet_work>

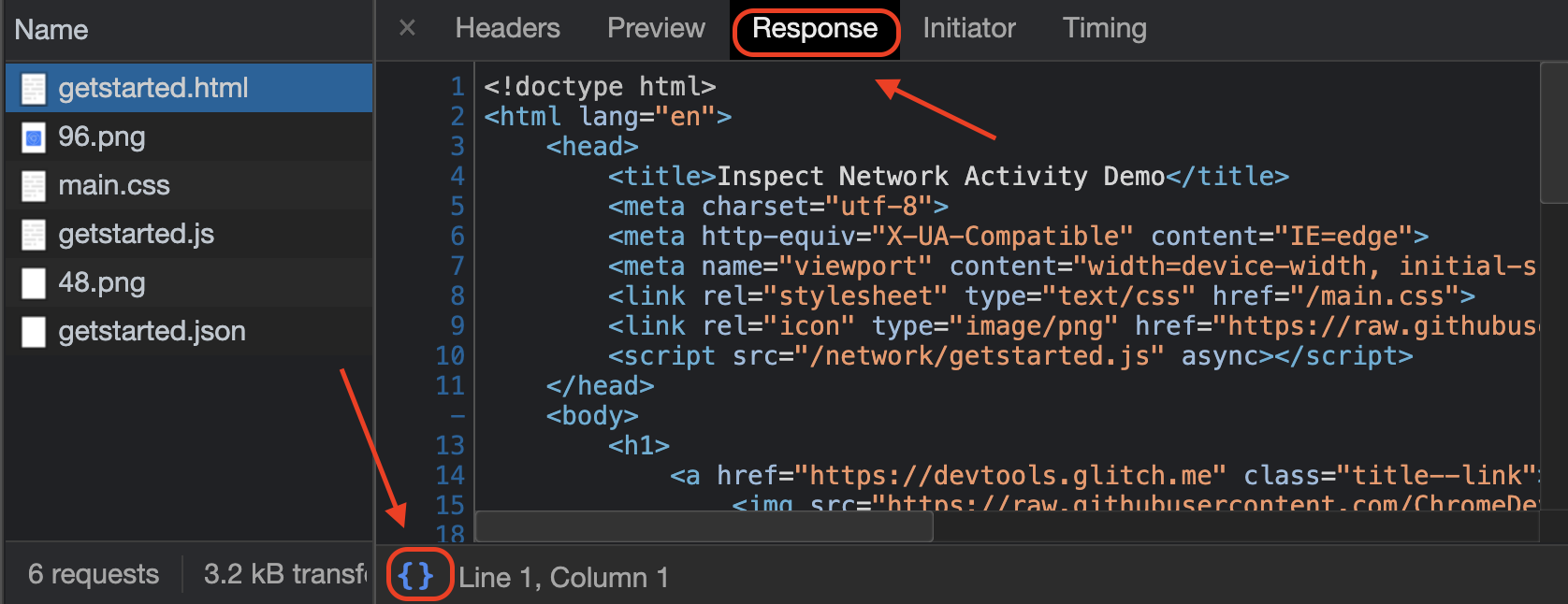
<https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works>

[Google Demo thing](https://developer.chrome.com/docs/devtools/network/#:~:text=Open%20the%20Network%20panel,-To%20get%20the&text=Open%20DevTools%20by%20pressing%20Control,The%20Console%20panel%20opens.&text=You%20might%20prefer%20to%20dock%20DevTools%20to%20the%20bottom%20of%20your%20window.&text=Click%20the%20Network%20tab.)

Talk about packet loss – some protocols allow for it (zoom), some don’t (downloading software)

TCP/IP organizes your packets

* Let’s just get the IP address for google using the terminal
* Run `ping google.com` (works on Mac and Windows)
  + This tells us the IP address of the server that is hosting the google homepage
  + It also tells us some things about our connection to that computer
  + Our computer communicates with the server computer by sending packets back and forth
  + Packets are pieces of data and they are how all data is sent over the internet
  + It tells us how much data it sent and how long it took and its time to live or hop count (the TTL number is the expiration if the request fails) (live like liv not like alive)
  + We should see something like 4 packets being sent and received, with hopefully no losses of data
  + Okay, so we’ve got the IP address now, let’s go test it out
* In your browser, enter the IP you got from pinging google -- 142.250.72.174 (or something similar, they can change)
  + You should get sent to the google homepage!
  + We completed the loop ourselves!
  + Note: not every website is accessible this way as some companies block the capability
* So this is how we are communicating with other computers using the internet, but how is the data actually being sent? **Draw the steps below**
  + Draw: an outline of the US roughly and put some stars on it for cities and draw lines between them, make sure there are multiple routes between places, shouldn’t be a single loop
  + Say: the internet is kind of like roads and railroads and even air space that we fly in -- it’s the connection between all the different computers that are hooked up to it. We send and receive mail and packages to each other that travel in cars or planes to get to other places. We send and receive data that travels in packets to get to other places.
  + Draw: a little stick figure in one of the cities and another stick figure in a far away city
  + Say: let’s pretend Person A has something that they want to send to Person B, but it’s been a while since they talked and so they need to get Person B’s address. They call a mutual friend and ask for Person B’s address. This is like accessing your ISP DNS lookup (Perhaps calling the person directing would be like getting all the way to the Root DNS while having their address written down in your contacts would be like having an IP in your machine’s local cache)
  + Draw: a pretty direct line between the two cities using the routes you made before, perhaps use a different color
  + Say: Person A tells the post office that they want to mail this letter to Person B and gives them the address they now know and the post office decides an efficient route for the letter to travel. That’s us asking the browser to send our request for a certain page out to wherever that page is hosted. Person B then receives the letter and wants to reply. They can see where the letter came from -- the return address. Now they write something back and tell the post office where to send the letter. That’s the server responding with the information we asked for. And the letter travels to Person A with the information from Person B.
  + You could also draw and explain at this point how things might have to go a different route -- in real like this might be due to weather, on the internet it could be a bottleneck or unavailable routes
* Git bash – ping google.com
* Okay, this is all very high level and big to comprehend, there’s a lot to it. Let’s take a look at a smaller example -- accessing things on our personal networks (your Wifi). Download the demo folder for today and in the terminal inside that folder run `npm i` (just tell them it’s installing things, don’t have to be super in depth):
  + Open up the folder provided in VS Code, we’ll be working with serverOne right now
  + Go through and explain what’s happening -- we’ll send an h1 tag when requests are made to this server
  + Start up the server by running `node serverOne.js` in the terminal (don’t have to use nodemon and they don’t have it yet, node is fine)
  + Go to the browser on your computer and go to localhost:4000 -- Devmountain rocks! Should be there -- remind them that localhost stands for 127.0.0.1, so we could actually change from localhost to 127.0.0.1:4000 and see the same page
  + So what this means is that you are currently running a server on your own computer and it can be accessed within your network! The computer can access itself, but what about other devices on your network? They can too! Let’s figure out our IP addresses and visit our little page from our phones. (Make sure your phone is on the same WiFi as your computer)
  + In your terminal, run this command
    - **Mac: `ipconfig getifaddr en0` (will just spit out a plain IP address)**
    - **Windows: `ipconfig` (gives a little more information, we want the IPv4 address)**
  + Now that you’ve got that address, you should be able to go to it at port 4000 - ###.###.###.###:4000 - in your phone browser and see the Devmountain rocks page
  + This is a small scale example of what’s happening with the entire internet -- some computers are only servers. They sit there with information waiting for other computers to ask for it. When they do, the server happily gives it to them.
  + Pretty cool, right? Let’s make some different types of requests now
  + In the terminal (Mac and Windows), we’ll use a command called curl to get some info
    - curl -i <http://www.google.com>
    - Scroll back up to the top of the request - We should get a bunch of info back, highlight the Content-Type, it’s HTML. You could talk about the Set-Cookie stuff too if you want to.
    - It was the i flag that allowed us to see the response headers (content-type, set-cookie, etc)
    - Then we’ve got a bunch of minified HTML - gross, but cool!
    - Curl is automatically using a GET request and it’s just going and getting the HTML that’s available at google.com because that’s what google has told that particular server to send along, let’s look at a smaller example
  + In your demo folder, open up serverTwo.js and take a look. Explain what’s going on -- we are getting different information depending on what type of request is made and where (the URL)
    - Run ‘node serverTwo.js` and go to localhost:4000 in the browser
    - Without any additional information after the slash, we’re just seeing that p tag
    - Let’s add /api/list -- we should see the info from the array!
    - Let’s try this in the command line now -- how could we use curl to see our information?
    - curl -i <http://localhost:4000> -- we should see the HTML that we sent back and it knows that content type is HTML and gives us some other info too
    - Now let’s try getting our array -- curl -i http:localhost:4000/api/list
    - Now the content type is json -- that notation that’s similar to JS, we can see in the server that we’re using JSON. That’s what we’ll be using to send information on servers in this course, not just plain JS. anyway, we’ve got our array and that’s pretty neat. So far we’ve only been making GET requests, but what about POST? With POST requests, we can create new information.
    - So if we wanted to add a name onto the end of that array, let’s look back at our code -- we have a way to run a POST request, but it requires something called a body -- this is an object that we can send with our request that contains the data that we want to use. It’ll be a JSON object and we can send it with curl
    - In your terminal, let’s run
    - curl -i -X POST -H ‘Content-type: application/json’ -d ‘{“name”: “Dave”)’ <http://localhost:4000/api/list>
    - The -i is showing us the response headers still, the -X followed by POST lets curl know that we want to make a POST request instead of GET, and then the -H is to set the header, which we then tell to expect some JSON. the flag d tells it we are about to give it some data, which we then do by sending a JSON object in a string. If we look at the server, our function was expecting a body on the request -- cool, that body is our JSON object -- and it was expecting the body to have a “name” property. That’s why we did name, and then we could give it whatever value we wanted -- Dave in this case
    - And in our response, we can see that Dave has been added to the end of our array!
    - In the web applications that you use every day, these requests are tied to URLs and user interactions -- when you press “send” on an email, that request is being handled by a server
* Now let’s get into the Network dev tools -- this will show us a bunch of information about the pages that we are interacting with on the internet.
  + Go to this link: <https://devtools.glitch.me/network/getstarted.html>
  + Open up the devtools and reload the page, you should see some files show up in the tab
  + We can see info about them like their status, type, how long they took to load -- and if we right click on the table header (like on Name or Status or whatever) then we can see lots more options, like the method, which will show us if it’s a GET, POST, etc, feel free to explore these with students
  + click on the file names and talk about them more -- we can see HTML, CSS, pictures, these all came from different requests made by this page (your devtools should be looking like the screenshot below now)
  + Make sure you’re in the Response tab -- you can click through the others to show different things
  + Also, if you click the empty curly braces, it will make code prettier



* Okay, now click the “Get Data” button on the page, and the gettingstarted.json should show up and now we can view that data, pretty cool
* Let’s go to a bigger website now -- say apple.com and view their network requests -- you can go through how long things take, go through the files, add different columns, go through the cycle again of request/response