**Web Client and Server Notes**

1. **Restaurant Analogy**
   1. The customer is the client
   2. The waiter is an API
   3. The cook is a Server
   4. The food is the data
2. **What is a Client?**
   1. Client programs connect to and communicate with servers, examples:
      1. A device, such as a phone
      2. An operating system
      3. A browser
      4. Something inside a browser
3. **What is a Server?**
   1. A computer program on standby, runs a loop looking for requests, when received - the program wakes up, processes the request, and generates a response, then goes back to standby
   2. Ex:
      1. A computer server (like Amazon Web Services)
      2. A server OS (like Windows Server Edition)
      3. A server program (Express server)
      4. Some part of a server program (my JS code listening for requests on a socket)
   3. **Listeners** are procedures or functions in a computer program that wait for an event to occur then respond - basically an infinite loop that reacts to inputs/events
      1. Ex: app.listen( 5000, ()=>{ console.log(`Listening on port 5000`); } );
         1. Enables the server to receive requests
   4. **Routes** take a request and provide a response of what to do
      1. Ex1: app.get( ‘/’, (request, response)=>{ response.send(`Hello world`); } );
         1. ‘/’ represents root path
         2. ‘request’ object contains information about the request itself
         3. ‘response’ object has many different methods (like .send() ) that help us deliver a response
         4. ‘.send()’ applied to a response will output the value we pass as plain text in the browser
      2. Ex2: req.params: { “userID”:”34”, “bookID”:”8989” }
         1. Route path: /users/:userID/books/:bookID
         2. Request URL: <http://localhost:5000/users/34/books/8989>
         3. Colon ( : ) is a placeholder, indicates a value to be queried
      3. Ex3: app.get( ‘/user/:name’, (request, response) => {

response.send(`Hello ${request.params.name}`); } );

* + - 1. This takes an input of the placeholder ‘name’ and feeds it into the response string
    1. Ex4: app.get( ‘/about’, (request, response)=>{

response.sendFile( `${\_\_dirname)/about.html`); } );

* + - 1. ‘sendFile’ is a method available to the response object
      2. Node variable ‘\_\_dirname’ returns a string with the directory the current file is in, useful when index.js and about.html are in the same directory
      3. This route sends the ‘about.html’ file when the ‘/about’ page is visited (a GET request) - the browser renders the html...
    1. A list of all methods in the ‘express’ library [can be found here](https://expressjs.com/en/4x/api.html#express.methods)

1. **What is HTTP?**
   1. A protocol - how a client talks to a web server
   2. The common language of the modern global Internet
   3. Not the only language (or markup) used on the global Internet
   4. Basic requests:
      1. GET - request data
      2. POST - give data
      3. PUT - change data
      4. DELETE - remove data
   5. Consistency with naming URLs
      1. Ex: A service that lists cars and drivers
         1. /cars - lists all cars
         2. /cars/2/drivers - lists all drivers for 2nd car
         3. /cars/4/drivers/5 - lists the 5th driver for the 4th car
2. **What is an API** (Application Programming Interface)?
   1. An Interface that determines how two programs should talk to each other
   2. HTTP is an API, it allows a program (like curl) to interact with a server vis HTTP methods
   3. REST is also an API that extends HTTP, has conventions about path names and how objects should be created, updated, and deleted
   4. Ex output of API output for my github account: <https://api.github.com/users/danbrierton>
   5. The web is an API
3. **What is DOM (Document Object Model)?**
4. **We will focus on software**
   1. A client program that initiates connections/sends requests to our server program
   2. A server program that accepts connections, receives requests, and returns a response to the client program
5. **Server-side tech**
   1. Note - ‘server-side’ means software code that runs on the ‘server side’ of the internet
   2. Express - a web application framework, server-side JS library
   3. Rails - a server-side Ruby library
   4. Sinatra
   5. Yarn - a package manager for JavaScript; downloads, installs, and manages project dependencies so you don’t have to (express and path, in our case)
   6. Heroku - allows users to deploy a server, live on the internet
6. **Client-side tech**
   1. curl
   2. Postman - an HTTP client program, used to test API’s you create for other Programmers, can send/view various HTTP requests (GET, POST, etc.) to servers and view responses
      1. Use dropdown to select HTTP methods and input an URL, run it
         1. Use the ‘Key’ and ‘Value’ sections to send data
         2. Use the ‘Raw’ dropdown to change the data type output
      2. Response is received in the ‘Body’ section
      3. Info about the received data (meta-data) is in the ‘Headers’ section
         1. Content-Type = the type of data received
         2. Status = a response summary with phrase & status code
   3. RequestBin - a website that allows you to view HTTP requests and details about them, like PostMan
      1. Useful for ‘POST’ing information from Postman
      2. Headers shows the meta-data
      3. Raw Body shows the data received
   4. Web Browsers
   5. AJAX
   6. React / Angular / Vue
7. **When an HTTP get request is made, is the CSS file also sent**? What about modules?
   1. Yes, CSS is sent to browsers, but possibly not to other things (like terminal)
8. **Creating a server with Yarn**
   1. Terminal: yarn check //validates you have yarn installed
   2. Make a directory, cd into it
   3. Terminal: yarn init; ls //creates your package.json file, verify
      1. Follow the prompts to name the server, version number, description, entry point, repository url, author, license, private
      2. Any prompt input can be blank by simply hitting enter
      3. MIT is the standard open-source license
   4. Terminal: yarn add express

//adds dependencies and creates node\_modules & yarn.lock

* 1. Terminal: touch index.js //or whatever path you put for ‘entry point’
  2. index.js: const express = require(‘express’):

const app = express();

app.listen(5000, ()=>{ console.log(‘Validate Server is Working’); });

//’app.listen’ is an example of a listener(see 3.c)

* 1. Terminal: node index.js //Validate it works
  2. Terminal: Ctrl + C to kill the server
  3. index.js: app.get( ‘/’, (request, response) => { response.send(‘Hello’) } );

// ‘/’ is the route (root), and request/response is ?

* 1. Browser: localhost:5000 //You should see ‘Hello’
  2. index.js: app.get(‘/user/:name’, (request, response)=>{ `Hello ${request.params.name}` ); })

// the colon (:) denotes a parameter to be saved into ‘name’

* 1. Browser: localhost:5000/user/dan //should display Hello dan
  2. Terminal: touch index.html; open index.html

1. **Deploy server to Heroku (make your stuff available in the www)**
   1. Declare an engine
      1. package.json: before the last curly bracket add:

“scripts”: {

“start”: “node index.js”

},

“engines”: {

“node”:”>=10.3.0”

}

* + 1. you need to add commas (,) to the {} contents outside the {}’s
    2. ‘scripts - start’ declaration tells the process that will run our app how to execute it (instructions of where to start)
    3. ‘engines - node’ declaration tells the process that will run our app what version of node should be used
  1. Add a dynamic port - because Heroku assigns an available port and we need to adjust accordingly
     1. index.js: const PORT = process.env.PORT || 5000;

//either heroku uses an environment port or 5000

app.listen(PORT, ()=>{ `Listening on port ${PORT}` );

});

* + 1. ‘|| 5000’ ensures that our app works both on Heroku and our localhost:5000
  1. Initialize a git repository and add .gitignore file
     1. Terminal: curl -o .gitignore <https://www.gitignore.io/api/node>

git init

git add .

git commit -m “add a message”

* + 1. ‘.gitignore’ is important to prevent access to files you don’t want to share
    2. ‘git add .’ adds all files in the current directory in queue to be commited
  1. Create the Heroku app and push it
     1. Terminal: heroku create dan-server

//make ‘dan-server’ whatever path you want it to be

git push heroku master

//this deploys the application

* + 1. Server is now deployed

1. **Creating a Simple Client**
   1. Create the DOM
      1. Create index.html and client.js
      2. index.html: <script src=”client.js”></script>
      3. client.js: document.addEventListener(“DOMContentLoaded”, function(){console.log(“Your DOM is loaded.”);});
      4. Validate the DOM is working by opening index.html in your browser, inspecting the page, select console, and look for the message
   2. Add AJAX request using XHR, change client.js to:



* 1. Add AJAX request using Fetch, change client.js to:



1. Add JQuery request:
   * 1. index.html, add script line:

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"> </script>

* + 1. Add ‘defer’ in the inline html for <script src="client.js" defer></script>
    2. client.js, change to:



* + 1. Validate it works by loading index.html in your browser, inspect the page, view the console

1. **DOM Manipulation** - how to make HTML dynamic
   1. Create dom.html, open it, load boilerplate
      1. Modify dom.html



* + 1. Validate changes by opening dom.html in browser
       1. Notice, ‘link’ should be displayed, but the script overrides it to say ‘Google’
       2. One could use ‘link.href=”new value”;’ to change the link destination
       3. Alternative to document.querySelector() are document.getElementByID() and document.getElementByClassName()

1. **Using jQuery for DOM Manipulation**
   1. Add the script tags to the above dom.html



* 1. Touch then edit app.js



* 1. Validate changes by refreshing dom.html in the browser (text should be red)

1. **API’s with jQuery**