Curriculum Review

Languages:

•Javascript - used for both front-end and back-end

•HTML – Hypertext Markup Language

• on websites, JSX (like in React)

•CSS/SCSS - (SASS) Cascading Style Sheets

• for styling/beautifying html sheets

•SQL - Structured Query Language

• used on database stuff to communicate with tables and stuff like that

•Regex - Regular Expressions

• identifying string patterns

Common Libraries:

•express - belongs in the server side; helps to communicate/to fetch from the server

•mocha - belongs in testing; it solves how to test things

•fs - file system; a node module that allows you to read and write things to files on your system

•passport - belongs in authentification and server side; implements sessions and solves the problem of distinguishing users

•socket.io - belongs to both client and server side; solves real-time bi-directional communication

•jQuery - belongs in client side; used to alter the DOM in js, simplifies client side scripting of html

•bootstrap - belong in CSS and front-end; similar to jQuery, a styling library made by Twitter, must be linked to HTML, ex// divides into containers

•sequelize - belongs in server side; javascript that communicate with your database

•nunjucks - belongs in front end/templating; required in express, connects in backend (four lines in app.use to configure the views in html)

•Google Maps API - belongs in the client side; gives us a script tag that comes with built in class names for styling, etc.

•async.js - probably belongs in the back end; utility module for handling asynchronous js

•bluebird - belongs in the client side; a promise library that needs to be imported (ex// .spread)

•react - belongs in client side; incorporates html into jsx, allows us to make components

•react-router - belongs in front end; url related, to utilize the routers ( like hash router, route paths, etc)

•redux - belongs in client side; state management, to create a store that holds all your states

•react-redux - belongs in client side; connects stuff, gets rid of subscribe and unsubscribe

•https - belongs in server side; secure http

Code Design:

•CRUD – Create (http - post) Read (http - get) Update (http - put, sequelize - update) Delete (http - delete, sequelize - destroy)

• defines basic functionality; only one can be used at a time

•MVC - Model View Controller

• implements user interfaces on computers, separation of view from state and both of those from user input, user uses the controller which manipulates the model and updates the view which the user sees (see wikipedia diagram)

•DRY – Do Not Repeat Yourself

• makes code cleaner, more concise

•SOLID - Single responsibility principle, Open/closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle

• to make software more understandable, flexible, and maintainable

•Dependency Injection - technique where one subject supplies of the dependencies of another object

• \*go look it up on wiki yourself

•ReST - Representational State Transfer

• simple stateless architecture, maps http requests to CRUD

Misc Terms:

•Library - means for programmers to develop dynamic interfaces, chunk of code you can call from within your own code

•Framework - group of libraries

•API - the functions/methods in a library, the interface to the library

•Protocol - common means for unrelated objects to communicate with each other (ex// HTTP, TCP, IP)

•Operating System - software that supports computer’s basic functions (ex// Windows, OS)

•Runtime Environment - a configuration of hardware and software, includes CPU type, operating system, and any runtime engines (ex// V8)

•Interpreter/engine/VM - virtual machine, pretend computer, interprets into a lower level language for the computer to process

Programming Fundamentals:

•Modularity - dividing shit up, subdividing a computer program into separate subprograms (ex// dividing into components)

•OOP – Object Oriented Programing; organized around objects rather than actions and data rather than logic (ex//js written object oriented-ly)

•Functional Programming - organized around functions, avoids changing-state and mutable data (ex// Redux reducer - pure functions, higher order functions)

•Recursion - when a thing is defined in terms of itself (ex// Fibonacci sequence)

•Closure - a part of scope, an inner function that has access to its own variables, the outer function’s variables, and global variables, but also binding

•Inheritance - when an object or class is based on another object or class using the same implementation (ex// superclass or subclass)

•First-class Functions - values in a language handled uniformly throughout, when you treat a function like a variable, all functions in js are first-class (can be used as a variable, can be passed as an argument, etc)

•Higher-order Functions - take or return a function, don’t manipulate the original data (ex// map or filter)

CS Fundamentals:

•Sorting – Bubble Sort and Merge Sort

• Merge: all the way down to individuals, then to pairs; Bubble: repeatedly swapping adjacent elements if they’re in the wrong order ( ex// largest bubbles to the top)

•Complexity - Space and Time

• represents worst case scenario the most space and time a program takes (ex// linear, exponential, log)

•Linked List - a data structure or collections of nodes, meaning a concrete programmatic way of managing information in memory,

• used to implement a number of ADTs, including Queues, Stacks, Lists

•Hash Table - a data structure used to implement an entity that allows dynamically storing and retrieving values via string keys, contains an array of "buckets" where it can store data

• set puts new data at the end, runtime complexity includes collisions

•Binary Search Tree - one for which every node has up to two children, a left and/or right child (binary) and when all nodes respect an order (tree): all values less than a given node value are in its left subtree, and all values greater or equal to a given node value are stored in its right subtree.

• \*whatever

•Cellular Automata - game of life, a collection of cells on a grid that evolve in time steps according to a set of rules

Javascript Fundamentals:

•Prototypes - instances of objects

•Types - boolean, null, undefined, number, string, and symbol

•Equality - == (loosely equal), === (deeply equal) (ex// ‘3’ == 3, true ‘3’ === 3, false)

•Context - What is ***this***? default binding rule, ***this*** references the global object; implicit binding rule, ***this*** references the object that the method is called on; explicit binding rule, ***this*** references .call, .apply, and .bind; ‘new’ binding rule, ***this*** references the new keyword

•Chaining - calling methods off each other (ex// array.map.filter.join() )

Dev Tools:

•Console - command-option-j; helps for debugging, and logging, and testing

•Inspector - looks at the elements on your page (also called Elements)

•Style Panel - lets you look at all the styles applied to a given element

•Network Tab - lets you see the status of your request (ex// 200, 404)

•debugger Statement - pauses your code so you can step through and gives information about variable values

Basic Node Usage:

•modules - libraries of code (ex// fs, http); implemented by import or require; ***module.exports***: the main one and ***exports***: reference to module.exports, used as short hand and cannot be reassigned

•npm - node package manager; helps you install modules

•package.json - a file that describes the project, lists dependencies, and defines scripts

Server-side Vs Client-side:

•dependencies - \*Emily/Ashi Tell us! Please!

•persistence - the continuance of an effect after its cause is removed, it outlives the process that created it

•Environment APIs - \*Emily/Ashi Help!

Asynchronous Programming:

•blocking v non-blocking - blocking is synchronous and non-blocking is asynchronous

•event handlers - ex// sockets.on(‘new message’, action) OR button.onClick()

•asynchronous callbacks - ex// readFile, setTimeOut

•promises - three states: pending, fulfilled, and rejected (ex// bluebird)

•middleware - what happens in the middle, what needs to be done before handling a request (ex// thunk, logging)

Promises:

•advantages - promises has error handling, and because callback hell is real

•.then - to do stuff with what the initial promises returns

•success and error bubbling - what is returned from then, needs to be handled by another then or catch handler depending on if its resolved or rejected

V8 Asynchronous Interals:

•event loop - pushes the callback into the stack

•call stack - blocking stack of actions being processed

•callback queue - before the event loop pushes into the stack, the list of callbacks that are waiting

•Environment APIs - where the async functions are handled, whose callbacks get moved into the queue

Network:

•TCP/IP - Explain TCP/IP and its realtionship to other subjects in this topic.

•HTTP - the communications standard driving the web. It uses the client-server model, where the server has resources which clients might want.

•web sockets - Explain Web Socket and its realtionship to other subjects in this topic.

•WebRTC - Describe WebRTC, comparing it to other subjects in this topic.

•AJAX - Describe AJAX, comparing it to browser-level HTTP.

HTTP:

•request – A URI path (uniform resource identifier) — a string label, for example ***/tweets/123***

•response – A status code indicating how the server dealt with the request (success, not found, etc.)

•request-response cycle - Define request-response cycle.

•verbs/methods - explaining what the client wants to do with the resource; some typical patterns: (GET: give me this resource, PUT: update the resource with this data, POST: create a resource using this data, DELETE: delete the resource)

•headers - Describe HTTP headers and list the common ones.

•Status Response Codes - Describe HTTP status response codes and list the common ones.

•cookies - What are cookies and what purpose do they serve?

Express Usage:

•Routes v Filepaths - How do routes and filepaths differ?

•app.use v app.all - What's the difference between ***app.use*** and ***app.all***?

•query, params, and body - Compare and contrast the request query, params, and body—provide examples.

•routers - What are express routers? How and why would you implement one?

•error handling - What is the standard approach to error handling in an express app? Provide code.

Common Middleware:

•morgan - a Node library logging middleware, which logs only upon response, merging in request info

•body-parser - Describe the role of body-parser in the middleware stack.

•express-session - Describe the role of express-session in the middleware stack.

•express.static - to serve static files such as images, CSS files, and JavaScript file

•node-sass-middlware - Describe the role of node-sass-middleware in the middleware stack.

Database Persistance:

•database - Define database.

•ACID - What does ACID stand for? What does each mean?

•relational v non-relational - Define relational v non-relational.

•CAP Theorem - What is the CAP thereom? What are its implications?

•SQL - Define and describe SQL.

•PostgreSQL - Define and describe PostgreSQL.

•schema - What is a schema?

•ORM – Define ORM.

•indexing - What is database indexing and why is it useful?

SQL:

•tables - Describe SQL tables.

•rows - Describe SQL rows.

•columns - Describe SQL columns.

•primary keys – Describe SQL primary keys.

•joining - What types of joins exist in SQL? How do they differ?

Sequelize:

•models - Define models and provide a code example. How do sequelize models differ from SQL schemas?

•validations - Describe sequelize validations and provide a code example.

•associations - Describe how sequelize enables relationships between tables and models. Provide a code example.

•eager loading - Define eager loading in sequelize and provide a code example.

•virtuals - Define sequelize virtuals and provide a code example.

•hooks - Define sequelize hooks and provide a code example.

•class v instance methods - Compare sequelize class methods and instance methods.

HTML:

•tag - Define tag in terms of HTML. Compare it with the others in this topic.

•attribute - Define attribute in terms of HTML. Compare it with the others in this topic.

•class - Define class in terms of HTML. Compare it with the others in this topic.

•id - Define id in terms of HTML. Compare it with the others in this topic.

•forms - How do forms work and how can they be configured to send HTTP requests?

(S)CSS:

•box model - What are the four 'areas' in the box model?

•inline v block - How do inline and block elements differ?

•specificity - When an element is styled by more than one rule, how does the browser determine specificity give precedence?

•grid layout - What is grid layout and how do you use it?

•float - What is 'float'? Give an example of where it would be useful.

•media queries - What problem do media queries solve and how?

jQuery:

•DOM Manipulation – Provide an example of DOM manipulation with jQuery code.

•DOM Traversal - Provide an example of DOM traversal with jQuery code.

•event handling - Provide an example of event handling with jQuery code.

•AJAX - Provide an example of an AJAX request with jQuery code.

•$ - What are the four roles $() can perform given different inputs?

React/Redux:

•react: components - What is a React component?

•react: jsx - What is JSX?

•react: state v props - What’s the difference between state and props? How do we change state? How do we pass props?

•react: lifecycle methods - Describe a component's lifecycle methods. When should you fetch data?

•react: stateless component - What is a stateless component?

•redux: store - What is a Redux store? How do you create one?

•redux: action creator - Describe an action creator versus an asynchronous action creator.

•redux: reducers - What is a reducer? How do you use ***combineReducers***?

•redux: dispatch - What does it mean to "dispatch an action”?

•react-redux - What does the ***connect*** method do, and how do we use it?