**Chapter 2:**

HTML = HyperText Markup Language

Documentation websites show how to use your code and what the methods do.

Head – Title goes on tab of website

Body goes on webpage

**Tags:**

<(HTML element open tag)> (content) </(close tag)>

<h#> = heading tag, get smaller as number increases, up to level 6 <h6>

**Self-closing tags:**

<br> = line break

<hr> = horizontal rule, adds horizontal line

**HTML attributes:**

<hr size = “3” no shade> size and noshade are HTML attributes

**Comments:**

<!-- (comments) -->

**Boilerplate:**

Reusable code templates (type html)

**Html document:**

<!DOCTYPE html> = shows what version of html we are using

<html> (content) </html> tells us everything in here is html

<head> (content) </head> holds information about webpage and how browser should handle webpage

Title - <title> (content) </title> title shown on browser tab

<meta charset=”utf-8”> = gives extra metadata to html doc

charset=”utf-8” = text inside webpage is encoded using utf-8 (8-bit Unicode transformation format) encoding system

different encoding systems are used for different languages (different characters)

utf-8 is the standard encoding system of html-5 as it includes all international symbols

name=”description” content=”(content)”

descriptions used by search engines when displaying results.

<body> (content) </ body >

Paragraph - <p> (content) </p> adds text in paragraph format

**Modifiers:**

Emphasis - <em> (content) </em> Italicizes the content. Also tells browser to stress the content

Italicize - <i> (content) </i> italicizes content. Only styles

Strong - <strong> (content) </b> bolds the content. Content has more importance

Bold - <b> (content) </b> bolds the content. Only styles

**Lists:**

<ul> (content) </ul> = unordered list

<ol> (content) </ol> = ordered list using numbers

<ol type=”i"> - i = lowercase Roman numerals, I = uppercase Roman numerals, a = LC letters, A = UC letters, default is numbers

<ol start=”4”> = starts using position 4 EVEN IF YOU ARE USING ROMAN NUMERALS/LETTERS

<ol reversed> = reverse order

<li> (bullet 1) </li> = bullet point of list

You can nest order lists in unordered lists and vice versa.

**Images:**

<img src=”(image source)” alt=”(description)”> - Inserts an image

Fetches image source from URL or local if in the same directory as website

Alt adds a description of the image if the browser cannot load the image properly

\*\*\*image attributes

**Hyperlinks:**

<a href=”(link)”> (link description) </a> = Anchor tag which adds a hyperlink with a descriptive text

You can link to a website or a local file in the same directory

\*\*\*anchor tag attributes

**Chapter 3:**

**HTML tables:**

<table> (content) </table>

<thead> (header content) </thead>

<tbody> (body content) </tbody>

<tfoot> (footer content) </tfoot>

<tr> (row content) </tr>

<td> (table cell content) </td>

<th> (header cell content) <th>

Cellspacing=”#” – pixel spacing between cells

Border=”#” – border between cells of # size

\*\*\*table attributes (mostly deprecated/not recommended)

**HTML forms:**

<form action="mailto:(emal) " method="post" enctype="text/plain">

<label>(content)</label> = creates label of name ‘content’

<input type="text" name="(name of content)" value=""> = creates text input field with a name

Can have text, color, checkbox, file, range, radio, date, password, email

Remove value for input type ‘submit’

\*\*\*forms/labels/input attributes

**Chapter 4:**

**Intro to CSS (cascading style sheets):**

Used to style whereas markup languages are used to program

<body style=”background-color: (content);”>

Style=”(CSS code);” = changes background color of the body portion of your html file using CSS code.

Can use default colors, #(hex values)

You can change styling in the head of your html document to affect everything in the body. If you have a specific change to an element in the body, it will overwrite the internal rule. The most specific changes are kept. Inline CSS – Same line; Internal CSS – in head tag; External CSS – External stylesheet.

Example:

<head> <style>

body { background-color: fuchsia; }

hr { border-style: dotted none none;

width: 5%;

border-width: 5px; }

</style> </head>

You can customize the global css by creating a css file and pasting the css code in it. Then using the link tag, you can link to that style sheet. The CSS file contains everything in the <style> tag.

<link rel="stylesheet" href="(css file locate)">

**Proper CSS Syntax:**

selector { property : value; }

who? { what? : how? }

selector = who or what is being modified (ex: <h1>)

Rules are in the brackets. They will change one property by giving it a new value.

Each rule ends with a semicolon. One rule per line. It’s good practice to organize alphabetically.

**Tags vs Class vs ID:**

You can style HTML elements (tags), classes or ID’s to differentiate between similar elements.

Tags are addressed as the tag name

Classes are addressed as .className

Classes can be used to group elements. Can have multiple classes per element.

Ex: <img class="bacon" src="(link)” alt=”bacon pic”>

.class{ Background-color: red; }

ID are addressed as #IDName

ID’s are unique. Can only have 1 ID per element. ID’s cannot be shared.

Ex: <h1 id="heading">I Love Bacon</h1>

#heading{ color: blue; }

ID > class > tag selectors in terms of precedence as ID is most specific

**Pseudo-class:**

Class assigned to elements with multiple states such as when an element is hovered over by the mouse

:(pseudo-element name)

Ex: .bacon**:hover** { background-color: green; }

**Chapter 5:**

**FavIcons: (favorite icon)**

Originally used in the favorites bar. Now displays on website tab

Create your own favicon at favicon.cc

Link to it with the link element

Ex: <link rel="icon" href="favicon.ico">

Browsers have their own style sheet called ‘user agent stylesheet’ which just give default styling values.

**Div element:**

The <div> element can group different elements together to structure your website more specifically.

<div class="topContainer"> <h1> I'm Mohanned </h1> <p> a cool guy </p> </div>

**The Box Model of website styling**

Different elements of a website are all boxes. Format the boxes to style your website.

**Block display elements:**

Paragraph (<p>), headers (<h1> to <h6>), divisions (<div>), lists and list items (<ol>, <ul>, <li>), forms (<form>)

Extend the entire width of the screen. Height is dependent on the content

You can edit parts of a block element using the <span> element

Ex: <p> a <span class="co">co</span>ol guy </p>

**In-line display elements:**

Take only as much space as needed to contain its elements

<img>, <span> and <a> (anchor)

Unable to adjust width of inline elements like you can for block elements

**In-line block display elements:**

Best of both world. Inline but can be adjusted similar to block elements (ex: width)

**None display elements:**

The element isn’t displayed at all.

This is different from the visibility selector as that makes your element invisible, but it still takes up space

**Rules of rendering elements:**

Content dictates size for inline elements

Order of elements is determined by placement in the code

Children elements are displayed over parent elements

**Positioning:**

Static – HTML element default position.

Relative – Position is changed relative to default position of element

4 coordinate properties (top, bottom, left, right) and value (px or %)

Element is moved away from the border direction stated in the amount stated

Ex: position: relative; left: 30px; moves image 30 pixels away from it’s default left border

This has no effect on other elements. Other elements will behave as if image was never moved

Absolute – Element position is changed relative to border of its parent element

This DOES affect the position of other elements. Absolute elements behave as if they are not there when determining placement of static or relative elements.

On the contrary, absolute elements are affected/pushed by static or relative elements.

Must have a parent which is relative otherwise absolute elements will compare to body.

Fixed – Stays in the same spot relative to website. It does not move when you scroll the page.

Useful for navigation bars or side bars.

**Centering elements:**

text-align: center; is placed in the parent element to center the contained elements.

Only centers elements that do not have specified width.

To center elements with specified widths, you must change margins (ex: margin: auto;)

**Font:**

font-family: (font 1), (backup1), (backup2);

You can embed fonts using google to ensure people can load the fonts you want.

<https://fonts.google.com/>

Add the embedded link in the head of your html file.

**Sizing:**

Font-size: (#)px; = changes size of font

#px is a static size. If user changes size of font, this will not change.

You can use % instead when specifying size to have text be dynamic.

You can also use the unit #em for dynamic scaling as well.

16px = 100% = 1em

You can change the font-size in the parent element and it will have multiplicate effects if you’re also changing font size in the child elements. This is not true for static sizing (#px). Static sizing is only affected by the most specific selector.

This can be worked around by using #rem which changes the root em. This is not affected by any upstream size changes such as in the parent element.

**Text wrapping:**

float: Allows images to float around text to have text wrap around an image.

Can float to left or right.

clear: Tells element to clear anything in direction stated. This forces text not to wrap around an image.

Can clear to left, right or both.

**Button generator website:**

<https://cssbuttoncreator.com/>

make sure to click beautify after implementing the button.

**Chapter 6:**

Intro to Bootstrap

Bootstrap is a Frontend framework/component library

It gives you preset classes which you can use by inserting the class name to your elements.

Responsive – adjusts to view port. Adaptable layout

CDN (content delivery network) = multiple areas to access content. Cuts down on latency to reach a website if it’s hosted in multiple locations. You will access the closest area.

You can load bootstrap onto your website by linking to the bootstrap CDN, by using the starter template from your CDN (easiest), or by copying the bootstrap CSS code into your website (slowest)

**How to create a website:**

Observe other websites for inspiration/ideas

Wireframe – low fidelity (just pen and paper quick sketch)

balsamiq.cloud – website to create wireframes

Mockup – high fidelity (what you see is what you get)

Prototype – create your website

**Columns** are divided in 12 units in bootstrap

You can specify how much units

<div class=”col-(#)”>

If you do not specify a unit #, it will divide the space evenly among your columns

You can specify unit numbers for different viewports

<div class=”col-md-2 col-sm-3 col-12”>

The smallest size column applies to all larger sizes unless otherwise specified.

If you have multiple CSS stylesheets applied, the one coded in LAST will be the one applied to your website. Order matters. Add the generic style sheet first then your more specific style sheets last.

**Class=”container” bootstrap element:**

Centers your container elements. container-fluid does not automatically adjust in sizes as viewport changes in size.

**Chapter 7:**

**Carousel:**

Can have automatic or manual carousels.

Can add buttons to move forward and backward

Can pause if mouse is hovered.

Can specify delay time before moving from one item to another.

Anchor tags can be used as buttons for forward and backward buttons.

**Card:**

Useful to display pricing (contains card-header, card-body, card-footer)

Wrap in card-deck for better formatting on 1 row.

Wrap instead with row/column to display better with different viewports

**Hierarchy:**

Element written last is applied last (final edit)

Element applied lower in code is higher up on Z-axis (more forward)

**Z-index** allows you to customize hierarchy

Only works with position relative, fixed or absolute

Higher numbers are placed further infront (more towards viewer)

Default z-index is 0.

**Media Query:**

Allows different CSS rules to apply given different changes to the media.

@media <type> <feature> {change something}

If the media is of type <type> of at least <feature> then {change something}

@media screen (min-width: 900px) {change something}

You can link to id or website.

**Code refactoring (tidy up your code):**

1. Readability
2. Modularity (organizing code in understandable sections)
3. Efficiency
4. Length (don’t sacrifice readability/modularity)

**Combining selectors:**

Multiple selectors can be selected by separating with a comma

H1, h2, h3, h4, h5, 6h {CSS code}

Can have space between commas.

Hierarchical selectors can be used to specify selectors given a certain condition

Selector1 selector2 {CSS code}

Selector1 is parent, selector2 is child. Must be separated with space

Combined selectors

Selector1.selector2 {CSS code}

Elements with both selector1 and selector2 are affected. Must not have space

**Ch. 8 - Web design**

**Color theory:**

Red – energy, love (cars)

Yellow – attention grabbing, intellect

Green – freshness, growth, safety (food stores)

Blue – stability, trust, serenity (cryptocurrency)

Purple – royalty, wealth, femininity (payday loans)

Analogous color palette - Two colors next to each other on color wheel, useful for logo and making harmonious colors. Does not stand out.

Complementary/clashing color palette: Use two colors from opposite ends of color wheel to make things stand out. Do not use this for text.

**Typography:**

Serif: letters have feet (horizontal lines) – more serious, authoritative, traditional, stable, respectable

Sans-serif: sensible, simple, straightforward

Stick to 2 or less fonts and keep to similar moods/time era

You can create interest by contrasting serif and sans-serif and the weights (thick and thin)

**User Interface (how you create your website):**

1. Hierarchy: Can be established by use of color (popping colors via contrasting with background)
   1. Use accent tones that are different from general theme
   2. Size (larger objects catch the eyes first)
2. Layout: mix things up with words and text to have website appear more interesting
   1. Keep text line length around 40-60 characters so its not too hard to read.
3. Alignment: Line up different items together to make things look more coherent (right aligned on same line)
4. White space: Use it, it’ll look better and less cluttered.
5. Audience: Keep audience in mind when designing a website.

**User experience (how the user feels when using your website):**

A good user experience won’t be noticed. A bad one will

1. Simplicity: keep things simple and clutter-free
2. Consistency: Be consistent between webpages
3. Use the F-layout: people normally read the first line then the left side. Keep important things on the left hand side.
   1. Alternatively use the Z-pattern
4. All platform design: Have a website that scales between the different platforms
5. Avoid evil practices or forcing people to do things they don’t want to (forcing people to turn off addblock)

**Ch. 9 - Javascript**

JS is interpreted language. It runs line by line.

JS is saved on your console between runs. You don’t need to repeat it on every run

Java is compiled language. It compiles the entire code first before running.

Function(“message”); <end

alert(“hello”);

**Data types:**

String: string A = “string”

Can be concatenated using a +

Numbers: 123

no keyword needed.

Boolean: boolean A = true or false

var Name = Data;

variable of name Name is equal to Data

variable name cannot begin with a number, be just a keyword or contain a space.

Only symbols that you can use are $ or \_

Camel-casing: lowerCaseThenUpperCase

**Functions - Strings:**

alert(message)

popup with message

prompt(message)

popup with message prompting a response

stringName.length

# of characters of string

First character is at position 0

stringName.slice(firstIndex, lastIndex)

returns string of firstIndex (inclusive) to lastIndex (exclusive)

toUpperCase()

changes all of the characters of a string to upper case.

toLowerCase() does the opposite.

**Functions – Numbers:**

+ - \* / basic arithmetic functions

% returns the remainder

8 % 3 = 2

Increment and Decrement

X = 5; X = X + 1; -or- X++; X = 5; X = X -1; -or- X--;

X += 2; X -= 2; X \*= 2; X /= 2;

Math.floor(number)

Truncates the number / rounds it down.

Math.round(number)

Rounds a number to closest whole integer

Math.pow(number1, number2)

Raises number1 to number2

Math.random() – generates random number from 0 (inclusive) to 1 (exclusive)

**Functions:**

Creating a function:

function functionName(arguement){

/\*function code

argument can be used here\*/

return answer;

}

Calling a function:

functionName(); //returns answer

if, if else, else (conditional) statements – control’s the flow of your code

comparators such as === or > or >= or < or <= allows use to compare two values

!== “not equal to”

=== checks if data type and value are the same

== checks only if value is the same. (number 1 is == to string 1)

Comparators can be combined with && (and) or || (or)

Arrays: var arrayName = [arrayElement1, arrayElement2, … ]

arrayName[#] – Retrieves element at position #

Indexes are counted beginning with 0

arrayName.length gives number of elements in array

arrayName.includes(item) – searches array for value item

arrayName.push(value) – adds value to end of array

arrayName.pop() – removes last element of array

arrayName.forEach(function(indexName){code here which includes indexName.key})

while (statement) loop: loops some code until the statement is no longer true

Used while something is in a certain state.

Code inside must ensure the statement become false at some point to avoid infinite loops

while (i >= 0){

code;

increment/change code;

}

for (start statement; end statement; incrementation/change) loop: loops some code with more specifications than a while loop until the end statement is no longer true.

Used to iterate a set amount of times.

for (var i = 0; i < 5; i++){

code;

}

target.addEventLister(“eventType”, listener);

Will call listener on target when eventType is reached

**You can use anonymous or named functions in addEventListener parameters.**

If your callback function has a parameter, you can call it using an anonymous function

target.addEventListener(“eventType”, function(){

myFuntion(argument);});

addEventListener is a higher order function as it takes functions as parameters.

listener is a callback function as it gets called back once an eventType is reached.

If you specify a parameter in your callback function, the event will be returned.

**Anonymous function:**

for (var i = 0; i < drumArray.length; i++) {

drumArray[i].addEventListener("click", function (){

alert("hey");

});}

**Lambda function:**

for (var i = 0; i < drumArray.length; i++) {

drumArray[i].addEventListener("click", handleClick);

}

function handleClick() {

alert("hey");

}

**Higher order functions can take functions as inputs.**

function add(num1, num2){

return num1 + num2;

}

function multiply(num1, num2){

return num1\*num2;

}

function calculator(num1, num2, operator){

return operator(num1, num2);

}

**Playing audio in JS:**

var audioName = new Audio(“audioFile.mp3”);

audioName.play();

**Objects:**

var objectName = {

property1: value1,

property2: value2,

property3: value3

method1: function(){

method1 code;

}};

Can have number, “string” or boolean values for properties

methods are functions that are associated with objects

objectName.property1;

calls property 1 of objectName

objectName.method1();

calls method1 of objectName

**Constructor function:**

function ConstructorName (parameter1, parameter2, parameter3){

this.property1 = parameter1;

this.property2 = parameter2;

this.property3 = parameter3;

this.method1 = function(){

method1 code

};

}

Var constructorObject1 = new ConstructorName(“value1”, value2, [value3.1, value3.2, …];

This creates constructorObject1 of type constructorName with the given parameters and attached methods

Constructor function names should be capitalized.

**Case, Switch, Break:**

switch (buttonInnerHTML) {

case "l":

var crash = new Audio("sounds/crash.mp3");

crash.play();

break;

default:

console.log(buttonInnerHTML);

break;

}

Works as a sort-of if, else if, else statement.

Checks the switch parameter, if it matches to the case value, executes the case code.

Break; discontinues the remaining switch code from running.

Default section is if there are no matching cases to the switch parameter.

**Ch. 11 – DOM**

You can have inline, internal and external JS similar to CSS. You want to apply a CSS style sheet in the beginning of your html document so the CSS styles are always applied. You want to apply your JS sheet later in the document so all the HTML and CSS files are loaded before the JS is applied.

Best practice is to place your JS sheet right before closing body tag of HTML document.

DOM (document object tree) formats your code into a tree. Similar level parts are siblings if they share the same parent. The head and body element are siblings who share the HTML parent.

Objects in DOM can have properties and methods.

Properties: describe the object (color of car, # of seats)

Can have get and set functions to obtain or change different properties

Don’t necessarily need () at end of these functions.

Car.color; (gets the color) Car.door(0); (sets the # of doors to 0

Difference between get and set is if there is a value in the ()

Whereas CSS-properties are usually hyphenated, JS properties are camelCased.

\*\*\***Values in JS must be written as strings (in quotes)**

Methods: things the object can do (drive, turn AC on)

Car.drive(); (method is called which makes the car drive)

Methods usually end with ()

**\*\*\*Method is something an object can do. Function can be used independent of an object**

**How to select different elements:**

querySelector(“tag/.Class/#ID”) – only returns 1 item (first in HTML code) with specified tag/Class/ID

querySelectorAll(“tag/.Class/#ID”) – returns an array of all items with the specified tag/Class/ID

firstElementChild.lastElementChild

getElementsByTagName(“tagName”) – returns array of all elements with tag name tagName

You can specify a certain index with getElementsByTagName(“tagName”)[specificIndex]

getElementsByClassName(“className”) – returns array of all elements with class name className

getElementByID(“iDName”) – returns one item with ID name iDName. Only returns 1 element as ID is unique. No brackets are needed when using this.

**Separation of concerns:**

It is good practice to have HTML used for content, CSS for styles and JS for behavior. In order to keep the separation of concerns true, you can use JS to change the classes of your elements and have different classes styled differently using your CSS. This prevents you from having to style your elements using JS.

.classList – displays the classes of a querySelector

.classList.add(“className”) – adds a class to the querySelector

.classList.remove(“className”) – removes a class from the querySelector

classList.toggle(“className”) – Adds a class if it’s not there/removes a class if it’s there from a querySelector.

**HTML - get and set functions:**

document.querySelector(“elementName”).innerHTML;

Returns the HTML of elementName.

This is more inclusive than textContent as it returns things like <b></b> or <i></i> as well.

You can change the HTML code by adding this to the end = “htmlCode”;

document.querySelector(“elementName”).textContent;

Returns only the text of elementName

document.querySelector(“elementName”).attributes;

Returns all attributes of given element.

document.querySelector(“elementName”).getAttribute(“attributeName”);

returns value of requested attribute of given element

document.querySelector(“elementName”).setAttribute(“attributeName”, “newValue”);

sets value of requested attribute of given element to newValue

**CSS – set functions:**

document.querySelector(“elementName”).style.cssStyleAttribute = “value”;

changes the cssAttribute to “value” of elementName

**Ch. 14 – JQuery**

JQuery is a JS library

Can be incorporated using a CDN (content delivery Network) or downloaded and added to website. CDN’s are faster, especially popular ones like Google’s JS CDN.

The script source must be placed before/above your own custom JS source near the end of your body tag. By placing your scripts at the end, you are ensuring they are fully loaded before being accessed.

If you want to place your script in your heading tag, you must include:

$(document).ready(function() { code placed here } );

This ensures your JQuery code is fully loaded before it is accessed.

**How to select elements:**

document.querySelector(“h1”) = JQuery(“h1”) = $(“h1”)

$ is short for JQuery

This searches for the element h1

document.querySelectorAll(“h1”) = JQuery(“h1”) = $(“h1”)

This also searches for all of the elements h1

If there are multiple elements that share the same element name/class/ID, using this search will return all of the elements which match. Using this to change a CSS property changes the CSS property for all of the elements that match as well.

**How to change CSS:**

$("elementName").css("cssProperty");

Returns value of elementName’s CSS property cssProperty

$("elementName").css("cssProperty", "cssValue");

Changes element elementName’s cssProperty to cssValue

$(“elementName”).addClass(“className1 className2 className3”);

Adds class className1, className2 and className3 to element elementName (no period needed before className1 2 or 3)

You can add multiple classes separated by a space.

$(“elementName”).removeClass(“className”);

removes class className to element elementName (no period needed before className)

$(“elementName”).hasClass(“className”);

Checks if elementName has the class className. Returns Boolean value (true/false)

**How to change HTML:**

$(“elementName”).text(“changedText”);

Changes the text of elementName to changedText

$(“elementName”).html(“changedHTML”);

Changes the inner html code of elementName to changedHTML

**How to manipulate attributes:**

$(“elementName”).attr(“src”);

Get function: Returns value of attribute src for element elementName

$(“elementName”).attr(“src”, “attributeValue”);

Set function: Changes value of attribute src to attributeValue for element elementName

**How to add event listeners:**

$(“elementName”).eventType(function(){

Anonymous function code;});

Adds an event listener of eventType to element elementName with the callback function in the parenthesis. Callback function can be anonymous or named.

**How to add or remove elements:**

$(“elementName”).before(“elementCode”);

Adds element elementCode before element elementName

Ex: $(“h1”).before(“<button>New Button</button>”);

Adds a button before your h1 element

.after can be used to add newElement after your specified element

.prepend(“newElement”);

Adds newElement after opening bracket of your selected element.

Ex: $(“h1”).prepend(“<button>new button</button>”); would become:

<h1> <button>new button</button> h1Content </h1>

.append(“newElement”);

Adds newElement before closing bracket of your selected element.

Ex: $(“h1”).append(“<button>new button</button>”); would become:

<h1> h1Content <button>new button</button> </h1>

$(“elementName”).remove();

Removes all targeted elements of elementName type.

**How to add animations:**

$(“elementName”).hide();

Hides the selected elementName element.

Alternatively could use .fadeOut();

Fades the element out gradually

$(“elementName”).show();

Shows the selected elementName element.

Alternatively could use .fadeIn();

Fades the elementin gradually

$(“elementName”).toggle();

Toggles the selected elementName element.

Alternatively could use .fadeToggle();

Toggles the element in/out gradually

$(“elementName”).slideUp();

Slides the elementName up.

$(“elementName”).slideDown();

Slides the elementName down.

$(“elementName”).slideToggle();

Toggles the elementName using the slide function

$(“elementName”).animate({JS CSS code here});

Will gradually change elementName to match the JS CSS code.

Only JS CSS code containing numeric values can be used. Things like color will not work.

\*\*\*These animations can be changed together to apply sequentially

Ex: $(“elementName”).slideUp().slideDown().animate({opacity: .5});

This will slide up, slide down and then change the opacity to 50% of elementName

**Ch 16. Unix command line:**

GUI (graphical user interface) is added to programs to allow the common user to understand how to use them. They hide the messy complicated coding side.

Command line interface: is just a window of code without any GUI present. CLI allows for greater control. It’s faster and more efficient than using programs with GUI’s.

**Different commands:**

~ directoryName$

Tells you what the current directory is.

ls

Lists all of the items in the current directory.

ls -a

Lists all of the items in the current directory, including hidden items.

cd C:\Users\Owner\Desktop\’Web Dev 101’\’2. Web Development’\

cd allows us to change the current directory

cd ~ changes the directory back to the root (computer)

cd .. takes you back one level

Alt + LMC – changes your cursor position to the desired area.

Ctrl + A – Moves cursor to the beginning of line

Ctrl + E – Moves cursor to the end of the line

Up arrow – cycles to previous line of code

Down arrow – cycles to next line of code

Ctrl + U – clears current line of code

mkdir directoryName

Creates new folder/directory with name directoryName in the current directory

touch fileName.fileExtension

Creates new file called fileName with extension fileExtension in the current directory

start fileName.fileExtension

Opens file called fileName with extension fileExtension in the current directory

start programName fileName.fileExtension

Opens file fileName with extension fileExtension using the program programName

rm fileName.fileExtension

Removes/deletes file called fileName with extension fileExtension in the current directory

rm \*

Removes all files in the current directory/folder

rm -r directoryName/

Removes the specified directory directoryName

pwd

Print working directory – prints out the current directory’s full path

**Back End (Node.js and Express.js):**

**Node.js:**

**Backend:**

Backend: Consists of server (provide html/css/JS), database (stores user data) and applications.

Division of front end and back end is useful to keep your code private and prevent theft. Front end is built for the common user and is easy to understand. With frontend only, all the code (HTML/CSS/JS) is sent from the server to the user’s web browser. By using backend, the calculations can be done server-side and only the results are sent back to the user’s browser. This helps conceal our code.

Back end languages have frameworks which allow for faster process of coding (Node.js > Express.js)

**Node.js**

With regular JS, we used javascript to interact with the users webpage. With Node.JS, we can now use javascript to interact with the user’s files and documents. Node.JS also lets us run the code on a server to allow for faster execution of code. The processed information is then transferred to the user to allow for increased speeds.

**REPL (read evaluation print loop)**

Allows us to run JS in CLI line by line.

Type in node in CLI to enter REPL mode.

You can exit node by hitting Ctrl+C twice or typing in .exit

To clear the console, type: clear

**Node methods in CLI:**

node JSFileName.js

runs JS file JSFileName.js in your CLI using Node.js

You can type in part of a method name and hit tab to autocomplete it or hit tab twice for a full list of all the methods that include that partial name.

**Node.js internal modules:** <https://nodejs.org/api/>

You can use internal Node modules to perform various JS functions.

One of the modules is the file system module. To gain access, you must first run this JS code:

const fs = require('fs');

This requests the ‘fs’ module and stores it in the constant fs

These modules are useful for accessing paths, opening, editing, creating, deleting or copying files.

**Node.js external modules:**

To create a package, type: npm init :into your CLI in the directory you want

Fill in the rest of the information to create your package.json file in the directory specified.

External modules can be installed from <https://www.npmjs.com/> (node package modules)

To request these files, you must run the file request line in your CLI in the directory you want.

Ex: npm install superheroes

These external modules will show up on the package.json file as dependencies.

You can now run the external module commands by first requiring the module and then using the module’s methods.

**Express.js:**

Express.js is a framework designed for Node.js to speed up coding and reduce repeated code.

To install Express.js, it is the same as installing an external module. Just initialize a NPM on the directory you want (npm init) using your CLI. Specify a JS entry point file. Then type: npm install express :in your CLI while in the same directory you want to save Express.js in.

To use Express.js, in your JS entry point, you need to write:

const express = require(‘express’); To request the express module

const app = express(); To save the express() function to app()

We can use Express.js to run our own locally hosted server.

App.listen(serverFrequency, function(){anonymous function code;});

This stars our server on <http://localhost:serverFrequency/>

This page is the root of our server.

Any code placed in the parameter is displayed on your CLI.

You can reset this server and its variables by typing in “rs” in your CLI

We can also change what is displayed on our web page

app.get(“location”, function(request, response){code;});

This Express method allows us to specify what is displayed on our webpage at the specified location (“/” is the root page).

The request parameter (req) lists a bunch of information on the user’s request

The response parameter (res) allows us to change what is displayed on our page

response.send(“HTMLCode”);

This sends HTML code to the webpage as a response. You can also send plain text.

response.write(“HTMLCode”);

cumulatively adds HTML lines to your webpage

response.end();

or can be substituted for response.send(); //empty parameter

response.sendFile(“filePath”)

Sends full files. You can send entire html files this way.

\_\_dirname

Returns the directory of the current file. 2 underscores!!

You can have multiple app.get methods with different route parameters to set a response to all of your different webpages.

You can use the nodemon npm to automatically refresh your servers

npm install -g nodemon

location it is saved does not matter

run server with nodemon using:

nodemon fileName.js

**How to access local images and css files:**

You can add local images and css files to your web page by using:

app.use(express.static("fileName"));

This allows you to access images and css files in the root folder fileName

**How to handle user input on server:**

Forms are used to contain user inputs (text, radio, submit)

<form action=”location” method=”post”> Input HTML code </form>

action=”/”

sends information to root page of server.

If no action attribute is specified, the form by default will send the info to the current webpage it is on.

The action attribute defines the location the data is submitted to.

The method attribute defines how the information is sent by the form

“get” sends data by the form although this data is visible in the page’s address field.

“post” sends the data by the form as well although the address field is not displayed.

After form data is submitted, your webpage must be able to handle a response.

app.post(“route”, function(req, res){code});

This is executed on submission of form data.

In order to utilize user data, you need to install the npm body-parser. This package parses incoming request bodies in a middleware available under the req.body property.

npm install body-parser

const bodyParser = require(“body-parser”);

app.use(bodyParser.parseType());

bodyParser.text()

parses code as plain text

bodyParser.json()

parses code as json file (similar to JS objects)

bodyParser.urlencoded({extended: true})

parses code from html form.

{extended: true} allows us to post nested objects.

req.body

creates the object body with properties and values from the form

Values are stored as strings. To convert to int, use Number(string);

**API’s:**

**Ch. 20 – APIs:**

APIs (application programming interface) are a set of commands, functions, protocols and objects that programmers can use to create software (such as with JQuery) or interact with an external system (such as tinder accessing Facebook user’s likes/music choices).

API components:

Endpoint – the starting url to reach an API

Path – One of the branches of the root (endpoint). These are planned for ahead of time when designing a website.

Parameters – Placed at the end of the url. Custom messages sent by user, not planned for ahead of time.

<https://urlName.net/pathName?key1Name=customQuery1&key2Name=customQuery2>

Parameters separated by path with ?. Each key-value-pair is separated with &

Order of parameters does not matter.

Authentication – Every time an API is accessed, the developer and the frequency of requests is tracked. If the same developer meets a certain threshold of requests, they will start being billed for using this API.

Searching for specific parameters of API’s can get messy so we can use the program Postman to declutter which parameters and values are being used when accessing an API.

Information from APIs can be returned in JSON format (JS object notation), XML (extensible markup language) or HTML (hypertext markup language).

JSON format is similar to JS object’s in notation, where each object can hold keys with values, separated by commas. JSON name’s are wrapped in quotes, JS objects are not.

You can use the chrome extension “JSON viewer awesome” to transform raw API data into readable JSON style.

**How to make an external HTTP request:**

https is a native module of npm.

const https = require("https"); //no installation needed since it is native\*\*\*

https.get(URLInQuotes, function(response) {

console.log(response);

});

This sends a request to URLInQuotes and console.log’s the response.

response.on(“data”, function(data){

JSON.parse(data);

//code goes here

});

This waits for data to be sent, then runs the following anonymous function.

The hexadecimal from your HTTPS request is parsed into JSON format.

Add your res.send(codeHere) in the code section to display a response after your HTTPS request and data is retrieved.

JSON.parse(stringHere);

Turns string stringHere to JSON format.

Websites normally return data in string format.

JSON.stringify(JSobject);

Turns JS object JSobject to compact string.

**How to post (send) data to external website:**

https.request(url, options, function(response){ code here; }

this makes a request to the specified web sever with the options specified with the callback function specified.

Options can include the method type (get, post), authentication type (appID), etc.

In order to send that data, you must save your request in a constant and then use that constant request to send the JSON data (in string format\*\*\*). Must finish with request.end();

const request = https.request(url, options, function(response){ code here; }

request.write(JSONdata);

request.end();

You can use this to add users to an email listing on mailchimp

By checking the statusCode of your request, you can check if it was successful or not (200 is successful, anything else may not be)

Mailchimp wants your data to be formatted as the following:

const data = {

members: [ {

email\_address: email,

status: "subscribed",

merge\_fields: {

FNAME: firstName,

LNAME: lastName

} } ] }

**How to redirect someone:**

If you have a button on your failure page which posts to a different page on your server (“/failure”), you can have a post request which redirects the user back to your root page (“/”). This works effectively as a reset button and will return the user back to the sign-up screen.

**How to host your website using Heroku:**

app.listen(process.env.PORT || 3000, function() { code here } );

process.env.PORT – specifies a dynamic port that Heroku will choose later or the port 3000 when running locally.

You must include a ‘Procfile’ file in your project directory which tells Heroku how to start your web server. ‘Procfile’ must not have an extension. The contents must only include:

web: node appName.js

In your CLI, you must do the following to add your file to git.

git init

initializes the git repository

git add .

adds all your files to the git repository

git commit -m “Message goes here”

commits the current version with the message.

heroku create

Creates the Heroku file.

git push heroku master

Pushes the current version to the Heroku servers.

If you want to update another version, you must do your git add ., git commit, git push heroku master again.

**Git and Github:**

Version control – Git allows us to save different versions of our program. We can rollback to a stable version if we made a change that breaks our code or continue to keep updating each version as we go.

**How to set up a local repository using git:**

git init

creates empty git repository in your current directory

git status

Shows you which files will be added to your staging area (intermediate area)

Non-committed files will be displayed as red

Committed files will be displayed in green

Modified files (from previous commit) will be displayed in red as well

git add fileName.extension

Adds a file to your staging area to be committed.

git add .

Adds all untracked files present in current directory (not previously committed)

git rm --cached -r .

Removes all files from the staging area.

git commit -m “message goes here”

Commits the files in the staging area as their own version

Write in present tense

git log

Displays a log of your commits, author, commit messages and time of commit.

Also displays unique commit hash which identifies each commit.

git checkout fileName.extension

Rolls back to the last version in our git repository

git diff fileName.extension

Shows you the difference between local file and last committed version

**How to ignore certain files from git:**

This is used for sensitive information (passwords/apiKeys) or user settings/preferences (DS\_Store)

Create your .gitignore file in your directory using your CLI

.touch .gitignore

Add the file names in your .gitignore file you do not want to commit

#Comment here

Allows you to put comments in your .gitignore file

\*.extension

This ignores all files with the specified extension from being committed

Github.com/github/gitignore

Contains many prebuilt .gitignore files you can use for your projects

**How to set up an external repository using github (git push):**

Set up your repository on github.com

In your CLI, type:

git remote add origin remoteRepositoryURL

This transfers your local repository to a remote repository you have on github

The name of your remote repository is usually kept as ‘origin’ (best practice)

git push -u origin master

pushes local repository to remote repository titled ‘origin’ on the master branch using the -u option.

Master branch is default/main branch of your repository

Master branch is your main branch of commits/save points. It is sequential. Main progress is saved here.

You can have a local repository in parallel with a remote repository. By synchronizing both files, you can keep both up to date.

**How to clone a remote repository onto a local repository (git clone):**

git clone remoteRepositoryURL

Clones the remote repository onto your local machine in the directory specified.

**Side branches:**

Side branches allow for multiple branches of the project to be worked on simultaneously. A side branch may be used to work on new features, bug fixes, experimental ideas, etc. You can make multiple side branches and you can also merge a side branch back into the main branch.

git branch branchName

Creates a side branch with name branchName

git branch

displays all current branches of your repository

The current repository is identified with an \*

git checkout branchName

Switches from current branch to side branch named branchName

**Forking (copying a remote repository onto your own remote repository):**

Once changes have been made on a forked remote repository, a pull request can be made from the original owner to the forked repository owner to incorporate/merge those changes onto the original remote repository.

\*\*\*

Git fetch updates your local repository with your remote repository

Git Merge merges your head branch with your remote branch/most up to date branch.

Git Pull fetches and merges your head branch all in one.

**Ch. 22 – EJS (Embedded JS templating):**

EJS allows us to update an HTML file we plan to send to our server based on given inputs. For example, we can display the day of the week before sending out HTML file instead of having to create 7 different HTML files.

**How to add values to your EJS file which replaces your HTML file:**

With EJS, you must have a folder called ‘views’ in your directory containing the ejs file you want to change. The EJS can be called whatever you want (list.ejs for example), although the extension will be .ejs and not .html.

After npm installing EJS, you must use EJS in your root JS file to implement it.

app.set("view engine", "ejs");

<h1>It's a <%= theDay %>!</h1>

Use <%= variableName %> to set up which variables you want changed in your EJS file

res.render('list', {theDay: currentDayString});

This updates the ejs file with the value currentDayString for the variable theDay

**How to create dynamic webpages:**

You can create dynamic webpages using:

app.get("/:webPageVariableName", func...)

To access webPageVariableName, use req.params.webPageVariableName

You can compare this variable with an array of current variables by using the npm module lodash. This module converts a string to it’s lower case version and removes any hypens, underscores or spaces.

**How to add JS to your HTML file:**

<% basic JS code here %>

Allows us to run basic JS code within the <% %> in your HTML file

Must include bracket and % on every line there’s JS code.

You cannot include any HTML in the brackets.

**How to reuse HTML code:**

In your ‘views’ folder, create a file containing the chunk of html code you want to reuse (ex: header.ejs). Write all of the html code you want inside of this file. In the html file you are rendering, include:

<%- include(“fileName”) -%>

The ejs extension should not be included.

You can use this to add headers and footers to keep your style between the webpages you render constant. This reduces the amount of reused code you must write.

**How to reuse JS code:**

In another JS file, write the code you wish to reuse (for example lets say we define 2 functions: getDate() and getDay()). Using module.exports (or just exports), we can now reuse the JS code in this file.

exports.getDate = function(){code goes here; return value;}

exports.getDay = function(){code goes here; return value;}

In the JS file we wish to reuse this code, you must first require this local JS file.

const JSobjName = require(\_\_dirname + “/JSfileName.js”);

const object = JSobjName.getDate();

**Scope** – The region of a computer program where the binding is valid.

A local variable (created in a function) will only be valid in that same function (it has local scope) and cannot be accessed outside of that function or in another function.

You can create var variables in for/while/if/else statements and they will still be accessible in the function that those initial statements are a part of.

Global variables are created on the outermost layer of a program (outside of any functions) and are accessible anywhere in the program, including any functions.

const values cannot be changes

var and let values can be changes

all 3 declared in a function are local variables.

All 3 declared outside of a function are global variables

var declared in a code block is global up to its direct parent (if code block is in a function, its global to its function, not globally)

\*\*\*Regarding const, you can still push values into arrays and change the values of object keys, although you cannot assign new arrays or new objects to constants.

**Databases, SQL and MongoDB, Mongoose**

Databases can be comprised of either:

SQL – structured query language. Old and reliable. More structured (requires a schema). “relational DB” (can have multiple tables in relation with each other to easier store relevant data and reduce repetition – customer table, product table, orders table). Scales vertically (not good).

Organizes data in tables like excel. Inserts null in empty data fields.

NoSQL – Not only structured query language. Includes SQL and another language. Tends to be newer. More flexible and not bound to a particular structure. “nonrelational DB” (better for 1 user: many posts, style). This type of DB allows for greater scalability as it uses a distributed system. Scales horizontally.

Data stored as JSON objects.

Top 4:

SQL: SQL MySQL, PostgreSQL

NoSQL: MongoDB and redis

**SQL:**

Main functions are (CRUD):

CREATE

CREATE TABLE table\_name (

column1 datatype,

column2 datatype… )

Creates a table with columns of specified datatypes.

INSERT INTO table\_name(column1, column2…)

VALUES(value1, value2…)

This inserts values into the table specified.

You can avoid naming which columns you want to insert data into by providing values for every column in the table

READ

SELECT column FROM table\_name WHERE condition;

This shows us the column of the table specified with the selected condition

SELECT \* selects all columns of the given table.

WHERE statement is optional

UPDATE

UPDATE table\_name

SET column1 = value1, column2 = value2…

WHERE condition;

This updates the values of the selected column of the selected table with the restricting condition. You can update multiple values at once.

ALTER TABLE table\_name

ADD column3 datatype;

Adds a new column column3 of the specified datatype

DROP COLUMN column2;

Deletes the specified column

ALTER COLUMN column1 newDatatype;

Changes the datatype of the specified column to newDatatype

DELETE

DELETE FROM table\_name

WHERE condition;

Deletes anything in the specified table with the specified condition

**How to link tables together:**

When you create a new table, you can add:

FOREIGN KEY (keyName) REFERENCES table\_name(primaryKeyName)

This links the newly created table’s keyName to the previous table\_name’s primary key primaryKeyName

If you want to create a new table with different columsn of different tables:

SELECT orders.order\_number, products.product, products.price, products.stock

FROM orders

INNER JOIN products ON orders.product\_id = products.id;

This combines different columns from different tables into one by combing the table orders with the table products. The values are identified by matching orders.product\_id with products.id.

**MongoDB:**

**How to create (databases, collections and documents):**

use databaseName

Creates a database with name databaseName, otherwise, switches to the DB specified.

db

Shows you which database you are currently in

show dbs

Lists all of the databases present

db.collectionName.insertOne({document})

collectionName is the name of your collection. If it does not exist, it is made.

Document comprises of fields and values. field: value, field2: value2, …

This is similar to key: value pairs of JS objects.

The first field is the \_id key which acts as a primary key for each document. It must be unique.

db.collection.insertMany({documents})

show collections

shows all of the collections in the given DB

**How to read:**

db.collectionName.find(query, projection)

searches the specified collection of the current db with the specified query and projection. Both arguments are optional.

db.collectionName.find(

{ age: { $gt: 18 } },

{ name: 1, address: 0 }

).limit(5)

Query restricts your search. Projection specifies which fields you would like displayed. 1 for projection means to display, 0 means do not display. For the example, this searches for all age fields with values over 18. It will display the names of those documents but not the addresses.

**How to update:**

db.collectionName.updateOne({\_id: 1}, {$set: {fieldName: value1}})

this updates \_id: 1 of collectionName to add fieldname: value1.

**How to delete:**

db.collectionName.deleteOne({filter: value})

This deletes the specified documents of the collection specified.

db.dropDatabase()

This deletes the current DB and its contents.

**How to establish relationships in Mongo:**

You can embed documents in other documents in an array for example

db.collectionName.insertOne({

\_id: 1,

Name: “Pencil”,

Price: 0.8,

Stock: 12,

Reviews: [{name: “Sally”, rating: 5, review: “best pencil”}, {name: “John”, rating: 3, review: “its alright”}]

})

A different way to create relationships is to have 2 separate documents in a collections (products), then, in a different collection (orders), reference the 2 documents you want to connect.

**How to use MongoDB with Node:**

Use mongoose…

**How to use Mongoose with Node:**

**How to create:**

const mongoose = require("mongoose");

//requires the Mongoose NPM

mongoose.connect("mongodb://localhost:27017/fruitsDB", { useNewUrlParser: true, useUnifiedTopology: true });

//Connects to the MongoDB

const fruitSchema = new mongoose.Schema({

name: String,

rating: Number,

review: String });

//creates schema for your collection with the specified fields and value types

//A document cannot include a field which is not included in the classification’s schema.

const Fruit = mongoose.model("Fruit", fruitSchema);

//creates your collection. Param1 = Name of collection (usually in singular tense).

//Mongoose will lowercase your collection name (using lodash) and pluralize it.

//Param2 = Which schema to use.

const fruit1 = new Fruit({

name: "John",

review: "lets see if this works",

rating: 640 });

//Creating a document of the specified collection

person.save();

//inserts your fruit document in your Fruit collection of your fruitsDB

Fruit.insertMany([fruit1, fruit2, fruit3], function(err){

if (err){ console.log(err); } else{ console.log("successfully saved all fruits to fruitsDB"); } });

//inserts your fruit documents in your array (param1) to the specified collection and DB.

//param2 is a callback function which can confirm successful or unsuccesful exection of code.

**How to read:**

Fruit.find({name: “John”}, function(err, fruits) {

if (err) { console.log(err); } else { fruits.forEach(function(fruit) { console.log(fruit.name); })

mongoose.connection.close(); } });

//searches your database and prints off the names field your documents with the given search criteria. By having an empty search criteria, it returns an array of all the documents.

//closes your connection to mongoose when it is done console logging the values.

**How to validate data:**

const fruitSchema = new mongoose.Schema({

name: { type: String,

required: [true, "whats the name??!!"] },

rating: { type: Number,

min: 1,

max: 10 },

review: String });

//This schema states a name is required and returns a custom error message when one is not provided. It also limits the rating from between 1 – 10, both inclusive.

**How to update:**

Fruit.updateOne({ \_id: "5f2c335482f60a28047141e4" }, { rating: 6, review: "I suck" },

function(err) { if (err) { console.log(err); } else { console.log("updated successfully"); } });

//updates the specified value of the document

//param1 = the search criteria. param2 = what values you want to update, param3 = callback function to handle errors.

**How to delete:**

Fruit.deleteOne({name: "Apple"},

function(err) { if (err) { console.log(err); } else { console.log("deleted apple successfully"); } });

//deletes the selected document

//param1 = search criteria. param2 = callback function to handle errors.

//you can substitute our deleteOne() for deleteMany(). It uses the same parameters.

**How to establish relationships between documents:**

const personSchema = new mongoose.Schema({ name: String,

age: Number,

favoriteFruit: fruitSchema })

//relationships. You can embed a document in the field of another document.

//If you save both documents simultaneously, the ID of the fruit will match the id of the Fruit in the person’s document.

**RESTful APIs:**

REpresentational

State

Transfer

You can make your API RESTful by using specific HTTP verbs and by using a specific pattern of routes/endpoint URL’s. This helps to standardize API’s amongst different websites so they follow a known set of rules and styles.

**HTTP verbs:**GET – Reads / gets the data of your server

POST – Creates / posts the data of your server

PUT – Updates the server / DB by sending an entire entry to replace the previous.

PATCH –Updates the server / DB by sending the specific entry you want changed.

DELETE – Deletes the specified data of your server.

**Specific pattern of routes/endpoint URLs:**

|  |  |  |
| --- | --- | --- |
| HTTPS verb | /articles | /articles/jack-bauer |
| GET | Gets all the articles | Gets the specific article |
| POST | Creates one article | --- |
| PUT | --- | Updates the specific article |
| PATCH | --- | Updates the specific article |
| DELETE | Deletes all the articles | Deletes the specific article |

By including all of these HTTPS verbs and route styles in your website, you have created a RESTful API.

You can create HTTP requests without populating the front end of your website through the use of postman. Use the form – url encoded option.

Use Robo 3T for a gui representation of your databases.

**Security and authentication:**

“Level 1” authentication (direct storage):

Store a username and password in your database. When the user logs in, check to see if there is a matching username and password.

“Level 2” authentication (encryption):

Caesar cipher – shifts the letters of your message by a certain number.

Use an encryption key to encrypt data. Data becomes encrypted when it’s saved and decrypted when it’s searched for (find()).

Use the module mongoose-encryption (AES) for a simple way to encrypt data using mongoose.

npm i mongoose-encryption

const encrypt = require("mongoose-encryption");

const secret = "thisisourlittlesecret"; //this is your encryption key. Keep this private.

userSchema.plugin(encrypt, { secret: secret, //this assigns your encryption key as the key.

encryptedFields: ['password'] }); //this chooses which fields to encrypt.

Use dotenv to store valuable keys and sensitive information. Ad the .env file to your .gitignore file so it is not committed and available to the public. When hosting on Heroku, it has a place to store all of your private keys and information which is kept offline.

require('dotenv').config(); //add this to the top of your code, as early in the commits as possible.

SECRET=thisisourlittlesecret //this is saved in your .env file in the same directory as your app

process.env.SECRET; //this allows you to access your .env variable.

“Level 3” authentication (hashing):

No encryption is needed as you don’t need to decrypt your data. This method is almost impossible to reverse. No encryption key is needed as well. If you run the same hashing algorithm on the same string, the same encrypted hash will be created.

npm i md5 //installs the md5 module in your directory.

const md5 = require('md5'); //requires the md5 module in the md5 constant.

md5(stringHere); //hashes the string inside the parenthesis

“level 4” authentication (salting)

Salting refers to adding a random set of characters to a user’s password before encryption. These set of random characters is referred to as salt, and they are stored in the database with the user’s username. This way, if two users have the same password, since they have different salts, they will have different encrypted passwords.

Bcrypt is a modern hashing algorithm which adds a random salt to user encrypted passwords. You can adjust the salt rounds (times the password is ran through the hashing algorithm) to make it more secure although this takes exponentially more processing power.

npm i bcrypt

const bcrypt = require("bcrypt");

const saltRounds = 10;

bcrypt.hash(req.body.password, saltRounds, function(err, hash) { password = hash; }

//this hashes a users password with a randomly generated salt

bcrypt.compare(passWord, foundUser.password, function(err, result) { returns true/false }

//this compares the user’s login password with the stored one.

“Level 5” authentication (cookies and sessions):

Cookies are stored locally and store previous session data so server’s can populate your web page with previous events (ex: cart on amazon.com).

Session: A type of cookie. The period when browser interacts with server. Starts when the user logs in and ends when they log out.

Passport is a npm which allows you to create cookies (more specifically sessions) and hash passwords.

npm i passport passport-local passport-local-mongoose express-session

const session = require("express-session");

const passport = require("passport");

const passportLocalMongoose = require("passport-local-mongoose");

Paste this below your app.set and app.use but above your mongoose.connect:

app.use(session({ //use sessions with the specific options

secret: "thisisourlittlesecret2",

resave: false,

saveUninitialized: false }));

app.use(passport.initialize()); //use and initialize passport

app.use(passport.session()); //use passport for sessions

Add these:

mongoose.set('useCreateIndex', true); //removes deprecation warning

userSchema.plugin(passportLocalMongoose); //used to hash/salt passwords. Saves users to DB

passport.use(User.createStrategy());

passport.serializeUser(User.serializeUser());

passport.deserializeUser(User.deserializeUser());

read about this…\*\*\*

“Level 6” authentication (third party OAuth):

OAuth is token based authorization

Why use OAuth:

1. Granular access levels – You can request specific things from the third party accessed.
2. Read / Read + Write access – Can retrieve or retrieve and change information from the third party.
3. Revoke access – The user can revoke access of the from the site in question from the third party site.

Set up site

Redirect to third party

User logs in

User grants access

Your site receives the authentication code (1 time use) or access token (much longer use; more information is accessible)

Read about this as well… \*\*\*

**React.js:**