Contents

[Unit 5-6 – CSS 2](#_Toc502574799)

[Unit 7 – Bootstrap 4](#_Toc502574800)

[Unit 8 – Intro to Javascript 6](#_Toc502574801)

[Unit 9 – Loops and Conditional Statements 7](#_Toc502574802)

[Unit 10 – Functions 9](#_Toc502574803)

[Unit 11 – Arrays 10](#_Toc502574804)

[Unit 12 – Objects 12](#_Toc502574805)

[Unit 13-14 – DOM 13](#_Toc502574806)

[Unit 16-17 – JQuery 16](#_Toc502574807)

[Unit 20 – Backend 19](#_Toc502574808)

[Unit 22 – Node JS and NPM 20](#_Toc502574809)

## Unit 5-6 – CSS

**Text and Fonts**

**Note:** not all fonts come standard in every OS, try to pick a commonly available one if not importing custom font directly.

**Common selectors**

**/\*font-family**

**use quotes for fonts with numbers\*/**

p {

font-family: "Arial";

}

h1 {

font-family: "Georgia";

}

**/\*font-size**

**use px or em-based on body px value as initial reference \*/**

body {

font-size: 10px;

}

h1 {

font-size: 5.0em;

}

p {

font-size: 2.0em;

}

span {

font-size: 2.0em;

}

**/\*font-weight**

**boldness or thickness of font. some options: bold, normal, 100-800 in steps of 100 \*/**

p {

font-weight: bold;

}

**line-height - line spacing, just like Word**

p {

line-height: 1.5;

}

**/\*text-align\*/**

h1 {

text-align: right;

}

p {

text-align: center;

}

**/\*text-decorations**

**used to give an underlying text effect like strikethrough\*/**

p {

text-decoration: underline;

}

h1 {

text-decoration: line-through;

}

**Box model**

From inside to outside: content, padding, border, margin

p {

**/\*Content-width and height**

**specify in px or percentage of parent element\*/**

/\*width: 200px;\*/

width: 50%;

/\*height: 300px;\*/

**/\*Padding**

**if just padding - same thickness all directions, otherwise specify orientation: left, right, etc\*/**

/\*padding: 10px;\*/

padding-left: 40px;

**/\*Border \*/**

border: 2px solid blue;

**/\*Margin –**

**if just margin - same thickness all directions, otherwise specify orientation: left, right, etc.**

**Similar to border, can specify all on one line \*/**

/\*margin: 100px;\*/

/\*margin-top: 500px;\*/

/\*margin: 20px 40px 500px 100px; \*/

/\*margin: 0 auto 0 auto;\*/

/\*first value sets top, bottom, 2nd value sets left, right\*/

margin: 0 auto;

}

## Unit 7 – Bootstrap

**Bootstrap -** Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. It can be either downloaded directly or linked with CDN.

**Images**

Thumbnails

<div class="thumbnail"><img src="https://images.unsplash.com/photo-1442406964439-e46ab8eff7c4?dpr=2&fit=crop&fm=jpg&h=825&q=50&w=1450">

</div>

**Buttons**

Ex:

<button class="btn btn-danger btn-xs">CLICK ME</button>

<button class="btn btn-success btn-xs active">CLICK ME</button>

<button class="btn btn-success btn-xs" disabled="disabled">CLICK

**Form Basics**

* **<div class="container"></div>** - makes items inside of it centered on page
* **<div class="form-group"></div>** - adds nice spacing for each control
* **class="form-control"** – custom form controls provided by Bootstrap library
* **<form class="form-inline"></form>** - formats entire form into inline version

**Navbars**

Navbars - responsive meta components that serve as navigation headers for your application or site. They begin collapsed (and are toggleable) in mobile views and become horizontal as the available viewport width increases.

**Supported content**

Navbars come with built-in support for a handful of sub-components. Choose from the following as needed:

* **.navbar-brand** for your company, product, or project name.
* **.navbar-nav** for a full-height and lightweight navigation (including support for dropdowns).
* **.navbar-toggler** for use with our collapse plugin and other [navigation toggling](https://v4-alpha.getbootstrap.com/components/navbar/#responsive-behaviors) behaviors.
* **.form-inline** for any form controls and actions.
* **.navbar-text** for adding vertically centered strings of text.
* **.collapse.navbar-collapse** for grouping and hiding navbar contents by a parent breakpoint.
* **.navbar-fixed-top** – makes the nav-bar stay at top of page while scrolling

Sample code

<nav class="navbar navbar-default">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-nav-demo" aria-expanded="false">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a href="http://www.google.com" class="navbar-brand">Koffee</a>

</div>

<div class="collapse navbar-collapse" id="bs-nav-demo">

<ul class="nav navbar-nav">

<li><a href="#">About</a></li>

<li><a href="#">Contact</a></li>

</ul>

<ul class="nav navbar-nav navbar-right">

<li><a href="#">Sign up</a></li>

<li><a href="#">Login</a></li>

</ul>

</div>

</div>

</nav>

**Grid system**

Bootstrap includes a responsive, mobile first fluid grid system that appropriately scales **up to 12 columns** as the device or viewport size increases.

**Column sizes:** col-xs, col-sm, col-md, col-lg

Syntax

<div class="row">

<div class="col-lg-6">1/2</div>

<div class="col-lg-6">2/2</div>

</div>

<div class="row">

<div class="col-lg-4 pink">1/3</div>

<div class="col-lg-4 pink">2/3</div>

<div class="col-lg-4 pink">3/3</div>

</div>

* Grid size references can be added together to control how the grid looks at each window size.

Ex: <div class="col-md-3 col-sm-6 pink"></div>

* Grid columns can be also nested.

Sample code (with nested grid columns)

<div class=”container”> /\*For centering entire grid on page\*/

<div class="row">

<div class="col-md-3 col-sm-6 pink">

<div class="row">

<div class="col-lg-6 ">FIRST HALF</div>

<div class="col-lg-6 ">OTHER HALF</div>

</div>

</div>

<div class="col-md-3 col-sm-6 ">TOUR DATE</div>

<div class="col-md-3 col-sm-6 ">TOUR DATE</div>

<div class="col-md-3 col-sm-6 ">

<div class="row">

<div class="col-lg-2 "></div>

<div class="col-lg-2 "></div>

<div class="col-lg-2 "></div>

<div class="col-lg-2 "></div>

<div class="col-lg-2 "></div>

<div class="col-lg-2 "></div>

</div>

</div>

</div> /\*row end tag\*/

</div> /\*container class end tag\*/

## Unit 8 – Intro to Javascript

Javascript is a **dynamically-typed** language. It can change a variable from one type to another.

Use **typeof** method to check data types.

**Primitive data types**

* **Numbers:** integer and decimal values, doesn’t matter if whole or fractional value. Ex: 4, 9.3, -10
* **Strings**: text values denoted **with single or double quotes**. Ex: “Hello World”, ‘Hello World’ , “43”,
* **Booleans:** binary value. Ex: true, false
* **Null:** variables explicitly defined as null
* **Undefined:** variables that are declared but not initialized

**Math operations:** JS follows the order of operations, including parenthesis

* sum: +, difference: -
* division: /, product: \*
* modulo(rem): %

**String operations**

* Concatenation: use + sign.

Ex: “charlie” + “brown” // “charliebrown”

* Escape characters start with “\”

Ex: “Singin \”Do wa diddy, diddy, dum diddy do\” “

Output: “Singin “Do wa diddy, diddy, dum diddy do””

“This is a backslash: \\”

Output: “This is a backslash: \”

* Strings have **length** property.

Ex: “hello world”.length // 11

* Access individual characters using **[ ]** and an **index**

Ex: “hello”[0] // ”h”

“hello”[4] // ”o”

* **Number(**string); //Converts string value to number data type

**Variables:** containers that store values. Generic type **var** can store all data types.

Syntax:

**var** variableName = yourValue;

**Comments**

Syntax: //This is a comment

**Useful built-in methods**

* **alert()**: displays a popup window with message.

Ex: alert(“This is an alert!”);

* **prompt()**: displays a popup window with input field asking for more info.

Ex: prompt(“What is your name?”);

* **console.log()**: displays a text message/string on the JS console
* **clear():** clears JS console from previous text

**Writing Javascript in a file:** use element <script src=”location”></script>

<head>

<title>Script Demo</title>

<script src="script.js"></script> </head>

## Unit 9 – Loops and Conditional Statements

**Basic conditional operators**: ==, >, < , >=, <=

**Special equality operators:**

* === : equal in value and type
* !== : not equal in value and type

**Comparison of == vs. ===**

var x = 99;

x == “99” //true

x === “99” //false, x is not a string

var y = null;

y == undefined //true

y ===undefined //false

**Special cases:**

true == “1” //true

false == 0 //true

null == undefined //true

NaN == NaN //false

**Values that aren’t actually true or false, are still inherently “truthy” or “falsey” when evaluated in a boolean context**

Falsy values: false, 0, “”, null, undefined, NaN

Truthy values: everything else

**JS Conditionals: if**(condition), **else if**(condition), **else**(condition)

Ex:

**if** (age < 18) {

console.log(“Sorry, you are not old enough to enter the venue”);

}

**else** {

console.log(“Come on in. You can drink”);

}

**Loops**

**While loop**

**while**(someCondition) {

//run some code

}

Ex:

//Version 1

**var** answer = prompt("Are we there yet?");

**while**( (answer !=="yes") && (answer !== "yeah"))

{

answer = prompt("Are we there yet?");

}

alert("YAY, WE MADE IT!!!");

**For loop**

**for**(init; condition; step)

{

//run some code

}

Ex:

console.log("PRINTING ALL EVEN BETWEEN 10 AND 40");

**for**(**var** i = 10; i <= 40; i += 1){

**if**(i % 2 === 0)

{

console.log(i);

}

}

## Unit 10 – Functions

**Function:** wraps sections of code into reusable packages

**Function declaration**

function doSomething()

{

console.log(“HELLO WORLD”);

}

**Function call**

doSomething();

**Arguments**

Ex: functionsquare(num)

{

console.log(num \* num);

}

**Return:** outputs value from function, stops function execution

Ex:

function capitalize(str)

{

return str.charAt(0).toUpperCase() + str.slice(1);

}

var city = “paris”; // “paris”

var capital = capitalize(city); // “Paris”

**Function declaration vs expression**

**Declaration**

function capitalize(str)

{

return str.charAt(0).toUpperCase() + str.slice(1);

}

**Expression**

Function declaration is lost if variable is redefined later

var capitalize = function(str)

{

return str.charAt(0).toUpperCase() + str.slice(1);

}

**Higher order functions**: take function as argument or returns a function

Ex:

function sing()

{

console.log(”twinkle twinkle…”);

console.log(“little star…”);

{

**setInterval**(sing, 1000) //Repeats sing function every 1 second

**Anonymous functions**

Declared function cannot be used outside higher order function call.

//Repeats sing function every 1 second

setInterval(function() {

console.log(”twinkle twinkle…”);

console.log(“little star…”);

{

, 1000)

## Unit 11 – Arrays

**Declaration**

* var friends = [ ];
* var friends = new Array(); //uncommon

**Initialization**

var num = [45, 37, 89, 24];

**Properties**

* arrays are indexed **starting at 0**
* can hold any type of data

Ex: var random\_collection = [49, true, “Hermione”];

* have a **length** property

Ex: var num = [45, 37, 89, 24];

nums.length //4

* Multi-dimensional arrays are possible

Ex:

var friendGroups =

[ [“Harry”, “Ron”],

[“Malfoy”, “Crabbe”],

[“Mooney, “Wormtail”] ];

**Adding to existing array by index**

var friends = [“Charlies, “Liz”, “David”, “Mattias”];

friends[length] = “Amelie”;

**Common array methods**

* **push(newValue)**: adds element to back of array, returns new array length

Ex:

var colors = [“red”, “orange”];

colors.push(“green”);

// [“red”, “orange”, “green”];

* **pop**(): removes element from back of array, returns removed element

Ex:

var colors = [“red”, “orange”, “green”];

colors.pop();

// [“red”, “orange”]

var col = colors.pop(); // col = orange

* **unshift(newValue)**: adds new element to array front. Similar to push but at front side
* **shift()**: removes element from array front, returns removed element. Similar to pop but at front side.
* **indexOf(searchItem)**: returns first index of searchItem in array. If element not found, returns -1
* **slice(startIdx, endIdx)**: copies parts of an array, doesn’t alter original array. Index endIdx is not inclusive. **If indexes not specified, slice copies the entire original array**

Ex:

var fruits = [‘Banana’, ‘Orange’, ‘Lemon’ ,’Apple’, ‘Mango’];

var citrus = fruits.slice(1,3);

//citrus now contains [‘Orange’, ‘Lemon’];

* **splice(startIdx, numElements)**: removes a quantity, numElements, of items from array, starting from startIdx index

**Array iteration:**  use a for loop or forEach() method in any array object

**forEach()**: built-in JS method of iterating over an array. Similar to range-based iteration in C++

Syntax: array.forEach(someFunction)

Ex:

var colors = [“red”, “orange”, “yellow”, “green”];

colors.forEach(function(color)

{

//color is a placeholder, can be called whatever you want

console.log(color);

});

## Unit 12 – Objects

**Initialization**

1. **Make an empty object and then add members to it**

var person = {};

person.name = “Travis”;

person.age = 21;

person.city = “LA”;

1. **All at once**

var person = {

name: “Cindy”,

age: 32,

city: “Missoula”

};

var person = new Object();

person.name = “Travis”;

person.age = 21;

person.city = “LA”;

**Retrieving data**

* Bracket notation

console.log(person[“name”]);

* Dot notation

console.log(person.name);

* Differences

Using bracket notation, one can look up a variable via the passed in string variable.Using dot notation is not possible if property starts with number or has space in it.

**Updating data**

person[“age”] += 1; //updates age for person object

person.city = “London”; //updates city for person object

Objects can be nested inside other objects

Arrays are a special subset of objects where the property/key is a number, array index

**Adding methods to Objects**

A method is a function that is added as a property to an object. It is called similarly to regular property of object, just with ().

Ex:

var obj = {

name: “Chuck”

age: 45,

friends: [“bob”, “tina”],

add: function(x, y)

{

return x + y;

}

}

Calling methods

obj.add(10,5); // result = 15

**this** keyword

Implicitly refers to the top-level object (that method is defined in). It can be used to access properties needed by the object’s methods

## Unit 13-14 – DOM

**DOM**, Document Object Model, is the interface between Javascript and HTML+CSS

**Applications:** games, scrolling effects, dropdown menus, form validations, interactivity, animations

**DOM Construction**

After loading an html page, the browser turns every HTML tag into Javascript object that we can manipulate. Everything is stored inside the **document** object.

**Object Model**: many Javascripts objects modeling info from HTML

**Document:** document stores these objects inside itself

**DOM Process**

SELECT and element and then MANIPULATE

**Selection**

<body>  
<h1>Hello</h1>  
<h1>Goodbye</h1>  
<ul>  
<li id="highlight">List Item 1</li>  
<li class="bolded">List Item 2</li>  
<li class="bolded">List Item 3</li>  
</ul>  
</body>

Ex: For our example, we'll change the <h1> color using JS

//Select h1, and save to variable

var h1 = document.querySelector(“h1”);

//Manipulating <h1> we selected

h1.style.color = “pink”;

**Methods for selecting elements**

* **document.getElementById():** takes a string argument and returns the one element with a matching ID
* **document.getElementsByClassName():** takes a string argument and returns (node) list of elements with a matching class name
* **document.getElementsByTagName**(): takes a string argument and returns (node) list of elements with a matching tag name
* **document.querySelector():** takes a string argument and returns the **first** element that matches a given CSS-style selector

Ex: var tag = document.querySelector(“#highlight”);

* **document.querySelectorAll():** takes a string argument and returns a (node) list of **all** elements that match a given CSS-style selector

**Manipulation**

**Changing an element’s style**

Basic Example:  
tag.style.color = "blue";  
tag.style.border = "10px solid red";  
tag.style.fontSize = "70px";  
tag.style.background = "yellow";  
tag.style.marginTop = "200px";

**Adding/removing classes**

Rather than directly manipulating style with JS, we can define a CSS classs and then toggle it on and off with JS

/\*DEFINE A CLASS IN CSS\*/  
.some-class {  
color: blue;  
border: 10px solid red;  
}

**var** tag = document.getElementById("highlight");  
//ADD THE NEW CLASS TO THE SELECTED ELEMENT  
tag.classList.add("some-class");

**Class methods**

//ADD A CLASS TO THE SELECTED ELEMENT  
tag.classList.add("another-class");  
//REMOVE A CLASS  
tag.classList.remove("another-class");  
//TOGGLE A CLASS  
tag.classList.toggle("another-class");

**Changing the content of a tag**

* **textContent:** returns a string of all text(pure text only) contained in a given element Property textContent is part of the element object.

Ex:

<p>This is an <strong>awesome</strong> paragraph</p>

//Select the <p> tag:

**var** tag = document.querySelector("p");

//Retrieve the textContent:  
tag.textContent //"This is an awesome paragraph"

//alter the textContent:  
tag.textContent = "blah blah blah";

* **innerHTML**: similar to textContent, except it returns a string of all HTML contained in a given element.

tag.innerHTML //"This is an <strong>awesome</strong> paragraph"

**Changing attributes (src, href, etc.)**

* **getAttribute(attribute):** returns attributes of an element

Ex:

<a href="www.google.com">I am a link</a>

**var** link = document.querySelector("a");  
link.getAttribute("href"); //www.google.com

* **setAttribute(attribute, newValue)**: sets or writes attributes of an element

Ex:

link.setAttribute("href","www.dogs.com");

//<a href="www.dogs.com">I am a link</a>

**DOM Events**

Events are everywhere

* Clicking on a button
* Hovering over a link
* Dragging and Dropping
* Pressing the Enter key

**Process**

We select an element and then add an **event listener.**

Ex: “Listen for a click on this <button>”

To add a listener, we use a method called ***addEventListener().***

Syntax:

element.addEventListener(type, functionToCall);

Ex:**var** button = document.querySelector("button");  
button.addEventListener("click", **function**() {  
console.log("SOMEONE CLICKED THE BUTTON!");  
});

**Common events**

* **click**: when clicking on something
* **change**: when data entered into text or number input or clicked on input
* **mouseover**:when hovering with mouse pointer over element
* **mouseout**:when moving mouse pointer away from element

## Unit 16-17 – JQuery

jQuery: a popular DOM manipulation library

**Common methods provided by jQuery**

* Select Elements
* Manipulate Elements
* Create Elements
* Add Event Listeners
* Animate Elements
* Add Effects
* Make HTTP Requests(AJAX)

**Why Use jQuery?**

* Fixes "broken" DOM API
* Brevity and Clarity
* Ease of use
* Cross-Browser Support
* AJAX

**Why Not Use jQuery?**

* The DOM API is no longer "broken"
* It doesn't do anything you can't do on your own
* It's an unnecessary dependency
* Performance
* Lots of people are moving away from jQuery!

**Including jQuery:**  can be done locally or remotely

<**script** type="text/javascript" src="jquery.js"></**script**>

<**script** type="text/javascript" src="https://code.jquery.com/jquery-2.1.4.js"></**script**>

**jQuery Selectors**

**Selecting Elements With $()**

Selecting with jQuery is very similar to **querySelectorAll**. We provide a CSS style selector and jQuery will return all matching elements.

Syntax: $(“selectorGoesHere”)

Ex: $(“img”) //to select all img tags

**Styling Elements Using .css() Method**

.css() method is jQuery’s interface to styling. It’s possible to select multiple elements with one selector and apply the same style to all of them.

Syntax: $(selector).css(property, value)

Ex: $(“#special”).css(“border”, “2px solid red”);

Passing in an styling info as object

**var** styles = { backgroundColor : "pink", fontWeight: "bold" };  
$("#special").css(styles);

**Common jQuery Methods**

* **val()**: returns and sets value of element(s), usually form elements
* Ex: $(“input:text”).val() //get value from a text input
* **text():** similar to textContent property in regular JS. Sets and returns text content for element(s)

Ex: $(“h1”).text(“new heading title”) //setting text

$(“h1”).text() //returning text

* **attr():** returns and sets the value of an attribute for element(s)

Ex: $(“img”).attr(“src”, “newURL”) //setting attribute

$(“img”).attr(“src”) //returning attribute

* **html()**: similar to innerHTML property in regular JS. Sets and returns html content for element(s)
* **remove()**: removes element object permanently from page
* **append(newElement)**: add new element object to page. newElement can be htmlString ,element, text node, array, or jQuery.

Ex: //adding new li element to ul parent

$("ul"). append("<li><span>X</span> "+ liText + "</li>");

Note: The three methods below can be chained together in one statement. Therefore, you can remove, add, and toggle multiple classes in one line.

* **addClass()**: similar to **classList.add()** method in regular JS
* **removeClass():** similar to **classList.remove()** method in regular JS
* **toggleClass():** similar to **classList.toggle()** method in regular JS

**jQuery Event Methods**

* Use **$(this)** to refer to element that had an event(that was clicked, keypressed on, etc)
* **click()**: a quick and easy way to add a click listener to element(s). **It will add listeners for existing elements only, compared to on().**

Ex: //Alerts when ANY button is clicked

$(“button”).click(function() {

alert(“Someone clicked a button”);

});

* **keypress()**: event sent when the browser registers keyboard input. It indicates which character was entered. If a user presses and holds a key, separate keypress events are triggered for each new character entered.

Ex: If lowercase “a” is entered, keypress will report 97, or “a”.

* **keyup/keydown()**:events sent when user releases or presses a key on keyboard. They provide a code indicating which key is pressed. This handler should be used for catching special keystrokes.

Ex: If lowercase “a” is entered, handlers will report 65, or “A”.

* **on()**: works similarly to **addEventListener()**. It lets you specify the type of event to listen to. **It will add listeners for all potential future elements.**

Ex: //alerts when ANY button is clicked  
$(“button”).on(“click”, **function**(){  
console.log("button clicked!");  
});

* **event.which**: for key and mouse events, this indicates the specific key or button that was pressed

**jQuery Effects**

**Fading**

* **fadeIn():** displays hidden elements by fading them into visibility
* **fadeOut():** hides matched elements by fading them out to transparent
* **fadeToggle():** fades element(s) in and out, similar to class toggle

Syntax: fadeOut(duration\_in\_ms, callback function)

Ex:

$('div').fadeOut(1000, function() {

console.log(“Element faded”) //console line executes after

}); fade completed

**Slide: adjusts height of element(s)**

* **slideDown():** displays matched elements with a sliding motion.
* **slideUp():** hides matched elements with a sliding motion
* **slideToggle():** displays or hides matched elements with a sliding motion

## Unit 20 – Backend

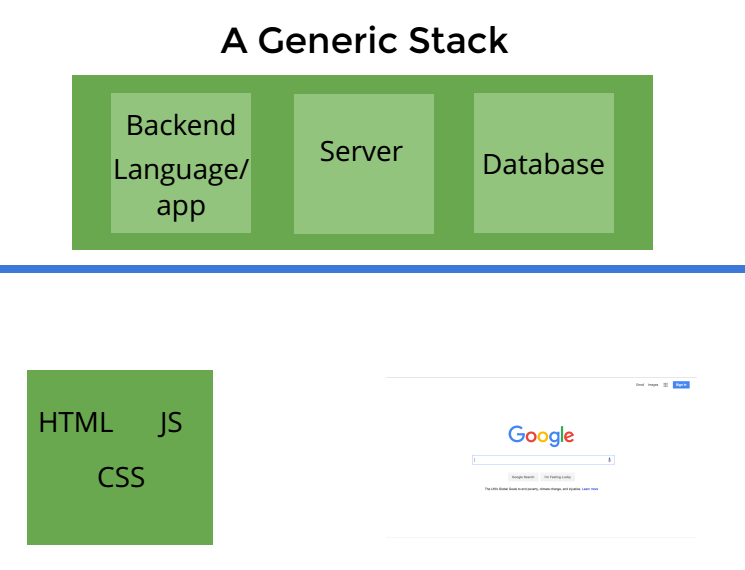
Loading a web page

* Entered URL is converted to IP address via DNS server.
* HTTP request is sent to server for the specific page with client IP.
* Server responds with a page with HTML, CSS, and Javascript

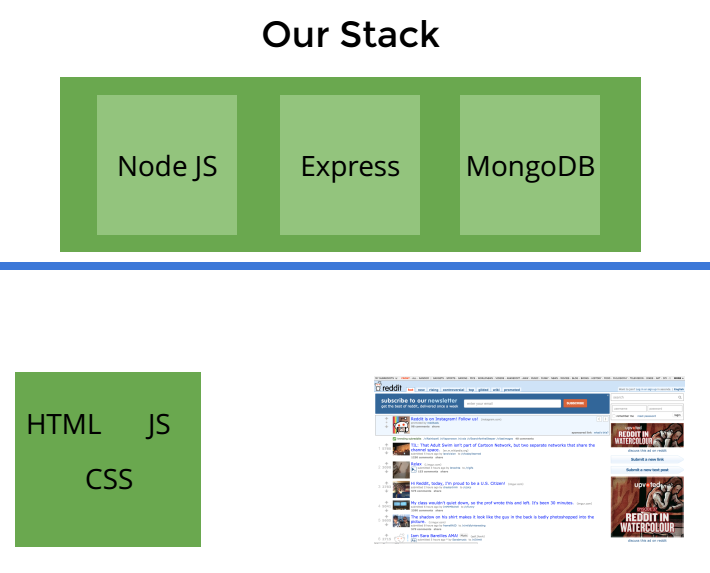
Static web page:

Dynamic web page: sites that are compiled on the server side. Server constructs a web page before sending it as a response.

**Generic stack**



Bootcamp stack



**HTTP Request types**

**GET**: getting or retrieving information from server

**POST:** adding/submitting information to server, ex: submitting form, uploading image, etc

Get Request example in Reddit search

URL: https://www.reddit.com/search?q=cats

Syntax: http://www.URL.com?variable=searchValue

Multiple variables can be added to the query by adding & between them

**Example of form submitting data via post request**

Note: **name attribute should be specified** to properly submit the data to the database to create the dogs object

<form action=”/createDog” method=”POST”>

<input type=”text” name=”name” placeholder=”name”>

<input type=”text” name=”breed” placeholder=”breed”>

<input type=”submit”>

</form>

## Unit 22 – Node JS and NPM

**What is Node.js?**

It’s a Javascript runtime built on Chrome’s V8 Javascript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js is how we can run JS on the server-side.

**Why are we learning it?**

* It’s popular
* Javascript
* It doesn’t matter(long term)

**NPM**

**NPM:** a package manager for Javascript, the largest manager of open source libraries in the world.

**Terminal commands**

npm init //create new package.json file

**Installing**

npm install **package** //local install(in app directory only)

npm install –g **package** //global install(in user folder, to be used by multiple programs)

npm install **package** --save //includes package as dependency

**package.json**: file holds various metadata information relevant to the project. It’s used to give info to npm that allow sit to identify the project as well as handle the project’s dependencies

--save //flag will take the package name and version and automatically save into the //package.json file

**Linking to npm package inside Node.js javascript**

Syntax:

var varName = require(“package”);

Ex:

var cat = require(“cat-me”) //package cat-me included

console.log(cat()); //Prints out via console.log picture of cat

## Unit 23-24 – Server-side Frameworks

**What is a framework?**

It’s a collection of code that someone else wrote that one can use. The control flow is on the framework side, not on the user side.

**How is it different from a library?**

The **inversion of control** is the difference. When calling a library, you’re in control and call it, if appropriate for your use case.

With a framework, the control is inverted, the framework calls you. Basically, all the control flow is already in the framework, and there’s just a bunch of predefine blanks spots that you can fill out with your code.

**Express framework**Currently, the most popular node.js web devframework. It’s a great tool for making web apps with.

Types of frameworks:

* heavy: does more of the work for you
* light: does less of the work for you, Express is one example

**Sample express code run with node.js**

//Importing module express

var express = require("express");

var app = express();

// "/" => "Hi there!"

app.get("/", function(req, res){

console.log("Someone made a request to /");

res.send("Hi There!"); //res is a response object

});

**Routing**

**Order of routes matters,** the \* matcher should be specified as last route, to allow other routes to be accessible. The logic for \* matcher route is similar to else statement.

“\*” route matcher //catch-all route that’s not specified specifically by other routes

**Route parameters (or path variables)**

Reddit example:

app.get("/r/:subredditName", function(req, res){

res.send("WELCOME TO A SUBREDDIT!");

});

You can access request parameters(params) from the url that was parsed as parameter initially.

Ex:

app.get("/r/:subredditName", function(req, res){

var subreddit = req.params.subredditName.toUpperCase();

res.send("WELCOME TO THE " + subreddit + " SUBREDDIT!");

});

**Templates and EJS**

You can use res.render() to render HTML (from an EJS file). EJS files should be located in **views** directory, as expected by Express

Ex: app.get("/", function(req, res){

res.render("home.ejs");

});

**EJS:** embedded javascript

**EJS syntax:** <%= %> //<===javascript code inside these tags

Passing object data from server to ejs file

app.get("/fallinlovewith/:thing", function(req, res){

var thing = req.params.thing;

res.render("love.ejs", {thingVar: thing}); //**thingVar:** variable specified in ejs file

});

**Dynamic HTML file:** a template (HTML file) that will change depending on data provided.