[S2:L13]

* DNS takes the domain name and convert it into IP address
* The request is made using the HTTP finding the server, hopping from one server to another.
* The server figures out what should it sent back, the server interacts with the database and pulls out the required data and respond with the combination of html, css, js.
* Browser then converts the response from server from a non-readable form to a human readable form.

[S2:L14]

* Front end is the stuff that you see and interact with: HTML, CSS, JS.
* Backend is everything other than front end.
* Frontend is what is platered and sent on the table if you go a restaurant and what happens in the kitchen is the backend.
* Dynamic page changes every sec because something changed at the backend when you refresh the page e.g. google news. Static page always stays the same.
* HTML is the skeleton of the page. CSS is the skin(flesh) to the HTML skeleton. JS is the functionality/interactivity of the page.

[S3:L24]

* <div> is a block level element whereas <span> is a in-line level element.

[S3:L25]

* The attributes take on a key-value pair.
* <img> is a self-closing tag
* <a> is a closing tag, if u wont mention “http” then it would consider the link as file protocol (relative path).

[S4:L32]

* We make a get request when we are asking for some information (doesn’t change the database) and we make a post request when we want to send that data to somewhere (adding something to the database).
* <form> takes an action (where it sends the input data) and the method (post/get), by default its GET request.

[S4:L37]

* Checkbox is used where we can pick multiple choices whereas in the radio button we only can pick one
* Select tag gives us the drop-down menu. We use it with the option tag.
* Name is like a key and value is the answer to that key.

[S5:L45]

* Hexadecimal color scheme: #FF0000 is red, ##00FF00 is green, #0000FF is blue.
* rgba color scheme we have 4 channels to fill out, where the 4th channel is for transparency (red 0🡪255, green 0🡪255, blue 0🡪255, alpha 0.0🡪1.0).

[S5:L46]

* background-repeat : no-repeat, doesn’t repeat the image even if the image is small to fit the page.
* background-size : cover, covers the entire page.
* A border has (width, style, color). None of them would work without the other, so would have to set all 3. We can set property individually or all in one line.

[S5:L48]

* An ID is a way to single out an element. We use ID to refer to special element.
* In CSS to select ID we use #, and for class we use ‘.’
* Text-decoration: line-through. Gives a line through a text.
* IDs are used for one element only, it can occur only once in the page.
* Class is just like an ID, but a same class name can be associated with different element, and can occur more than once on a page.

[S5:L49]

* Google inspector is a great tool to inspect elements of different site and play with it.

[S5:L50]

* Types of selectors:

1. Element, e.g. li {};
2. Class, e.g. .birth{};
3. ID e.g. #age{};
4. Star e.g. \*{};, changes applies to everything in the html.
5. Descendant, it takes at least 2 element selectors that are nested i.e. ancestor and descendant of each other e.g. ul li {};
6. Adjacent selector takes 2 siblings that are not nested within each other e.g. h4 + ul {};
7. Attribute selector: selects all the elements with a particular attribute e.g. a[href = “http://google.com”]{}; or input[type = “checkbox”]{};
8. Nth of type, selects every nth values of a type e.g. ul:nth-of-type(3){}; it selects the third one from every group of ul. Ul:nth-of-type(even){} selects every other even number in the group of ul.

[S5:L52]

* Inheritance in CSS, all the children inherit the properties of the parent. E.g. changing the color of body changes color of everything to that color.
* The selectors are assigned a numeric value to calculate the specificity.
* Selectors in terms of specificity: in-line > ID’s > Class > type (e.g. element , descendant, adjacent).

[S6:L57]

* Cssfontstack.com tells what % of OS supports different fonts.
* Font-family, gives the types of font.
* Font-size, controls the size of the font.
* If u r not sure what font size to use, then you can inspect it in the browser.
* Ems are used to specify the size relative to its parent size without specifying the size for itself, e.g. div {font-size : 200px;} p{font-size : 2.0 em;}, so with this the font-size of the paragraph would be twice the font-size of div its contained in.

[S6:L58]

* Font-weight controls the intensity of the font.
* Line-height controls the spacing between the lines.
* Text-align: aligns the text to either left, right or center.
* Text-decoration: gives underline, line-through to a text.

[S6:L61]

* Every element has a box around it.
* When we create a border, it extends to all the way to the end of the screen. It can be controlled by using the width property. Its better to use width in % because its relative to the parent width.
* A box has 4 parts. (contents, padding, border, margin).
* Between the text and border there is padding.
* outside of the border is the margin, it adds spacing between the elements.
* You can set border, margin, padding for left, right, top and bottom.

[S6:L65]

* To form a grid i.e. making one item appear after another, use float : left; it also takes out the space between different elements.

[S7:L74]

* Bootstrap is just an HTML, CSS and JS file that you can include in your code without knowing how its done.
* Without bootstrap we would need dozen of lines of html and css to achieve the same thing.
* Don’t be an expert in memorization, be an expert in accessing that information.

[S7:L76]

* Class “Jumbotron” takes the entire width of whatever container we place it in.
* Important classes

1. Form-group, it adds some nice spacing between different elements of the form.
2. Form-control, makes the input boot strapified.
3. Form-inline, all the form goes in nice little one line.

[S7:L77]

* <nav> adds the navigation bar.
* For nav bar hamburger button to show the drop down menu u need to import bootstrap script and jquery script.

[S7:L79]

* Main reason to use bootstrap is because of its grid system.
* For grid system, u need to use div class= “row” to specify the row and class = “col-lg-1” to specify a column. U can have 12 columns per row. We can nest
* Grid system divides the screen into pieces and we can specify how much should each piece take.
* Grid has 12 units that we can designate to different component in the container by class = “col-lg-1”.
* We have 4 sizes (lg, md, sm, xs)Col-lg-1 means that when the screen is at the large size then the component should take 1 of the 12 units. Similarly col-md-6 means when the screen at medium size then the component should take 6 of the 12 units.
* With class=”container” everything goes in the middle.

[S7:L82]

* To scale the image down to the size of the component in the grid use class = “thumbnail”
* Span is in-line and div in in-block.
* To make the navbar stick to the top even when scrolling use class = “navbar-fixed-top”
* To add icons, use glyphicon.
* The width of the navbar is 50px.
* FontAwesome has a huge collection of icons even more than bootstrap glyphicons.
* With tag html{height: 100%} the image covers the whole screen when the size of the window reduces.
* To give a shadow effect use rgba(0,0,0,0.3).
* For background images
* background-position: center;
* background-size: cover;
* background-repeat: no-repeat;
* html{height: 100%}.
* Or alternatively, use bootstrap class “img-fluid”

[S8:L88]

* Software release stages: alpha 🡪beta 🡪 stable
* Changes from V3 to V4
* V4 has the flexbox, v3 doesn’t.
* unit change from px to rem.
* We need to include popper, js, jquery cdn.
* Font-size increased from 14px to 16px.
* The nav, grid, cards now come with flex box.
* Thumbnails, panels are now gone, and we have cards.

[S8:L94]

* “Blockquotes” are used for citation.
* There are a lot more border classes and shapes and colouring for the border.
* Margin and paddings are based on rem rather than px.
* For margin: (m)(where: t, b, l, r, x, y )-(what size: 0🡪5), m-0 means that 0 margin on all sides and mb-5 gives margin on bottom of 5 units relative to a spacer element in bootstrap, mx-5 gives left and right a margin 5 and my-0 removes margin from top and bottom.
* For padding: (p)(where)-(what size), p-0 means that 0 padding on all sides and pb-5 gives padding on bottom of 5 units relative to a spacer element in bootstrap, px-5 gives left and right a padding 5 and py-0 removes padding from top and bottom.

[S8:L96]

* There are five display sizes: xs, s, m, l, xl.
* We have 5 display sizes in V4.
* To align the text in the center use class text-center.
* Breakpoints:
* Format for xs:- (m/p)( t, b, l, r, x, y)-(0🡪5)
* Format other than xs: (m/p)( t, b, l, r, x, y)-(s, m, l, xl)-(0🡪5)
* Breakpoints allows specifying padding/margin on different screen sizes for different sides, e.g. p-s-5 means that apply a padding of 5 on all sides when screen size small or above. NOTE: for extra small we don’t specify ‘xs’ we just say p-0 which means that apply padding 0 on all sides for screen size xs and up.

[S8:L97]

* Unlike V3, In V4 we have the control of when to expand the navbar at different breakpoints e.g. as sm, md, lg, xl and we can also give it a background e.g. bg-dark would give a dark background.
* Class = “navbar navbar-expand-xl navbar-dark bg-info” would give a navbar with light text and a blue background and would expand only when the display size is xl or above.

[S8:L98]

* Display utilities: -
* Format for xs: d-(value e.g. none, inline, block, table etc.).
* Format other than xs: d-(breakpoint i.e. sm, md, lg, xl)-(value e.g. none, inline, block, table, flex etc.).
* it allows us to hide something at some breakpoint e.g. <h1 class= “d-xl-none”>HELLO</h1>, would hide HELLO when the breakpoint reaches extra large.
* <h1 class= “d-none d-lg-block d-xl-none”>HELLO</h1>, would hide HELLO on every break point except at large.

[S9:L99]

* Flexbox allows us to position different elements in the container it’s contained in pretty easily, it also changes it height accordingly.
* Everything must be in the class “d-flex”
* For the x-axis (right 🡪left):
* Class Format: justify-content-(breakpoint i.e. sm, md, lg, xl)-(value i.e. start(by-default), end, center, between, around)
* Justify-content-start is by default which goes from left🡪right, justify-content-end makes the contents appear from right🡪left.
* Justify-content-between puts the element in the extreme ends or with the most spacing between them.
* Justify-content-around puts space between and around the contents.
* For the cross-axis (top 🡪 bottom):
* Class Format: align-items-(breakpoint i.e. sm, md, lg, xl)-(value i.e. start, end, center, baseline, stretch(by-default) )
* You can also align each item individually by replacing the ‘item’ in format with ‘self’. There is no justify-self

[S9:L100]

* To allow the contents take up the full size in the container its contained in weither horizontally or vertically:
* Format for xs: flex-(column/row).
* Format other than xs: flex-(breakpoint i.e. sm, md, lg, xl)-(column/row).
* Flex-column would give us the items in the flex in rows stretching horizontally, it automatically increases the height. Flex-column-reverse would give us item in rows but now the order is reverse from bottom🡪top. Similarly, we can add breakpoints for different view on different screen sizes.
* Flex-row would give us the items in the flex in columns stretching vertically. Flex-row-reverse would reverse the order. Similarly, we can add breakpoints for different view on different screen sizes.

[S9:L101]

* Because the navbar comes with flexbox, we can apply all flex properties like justify, align, flex-(column/row) to the navbar items.
* We can add breakpoints to make the navbar look different at different screen-size.

[S9:L102]

* In V3, the xs was less than equal to 768px but now in V4 xs is less than equal to 576px. so sizes have shrink down a bit.
* In V3 we specifies ‘xs’ size but now like padding, margin and other stuff we don’t specify ‘xs’.
* In both V3 and V4 we have 12 columns.
* The grid, like nav, is also built with flexbox.

[S9:L104]

* We can use flex utilities with cards.
* A card can have an images, header, footer, buttons, text, blockquotes etc.

[S9:L108]

* “aria-label” property helps defining a label for the buttons so it could give a meaningful explanation when using screen reader.
* To make the nav-bar stick to the top even on scroll use class = “fixed-top”

[S9:L109]

* “img-fluid” class automatically scales the image on different display sizes.
* Media query allows us to change the appearance of our website(e.g. the text size, background etc) when it reaches some screen size.
* Bootstrap class ‘order’ deals with which content should be shown first and which later e.g. order-1 means that the content should always be shown first whereas order-md-2 means that for screen size medium and above show it on the second place.
* without fixed-top the items would appear after nav border but within the items appear on the nav itself

[S10:L113]

* we can write the JavaScript in the chrome console.

[S10:L114]

* JavaScript has 5 primitive data types
* Numbers, weither positive, negative, float double etc.
* strings, anything inside either double or single quotes.
* boolean, either true or false.
* Null
* Undefined.
* JS escape character ‘\’.
* You can use single quotes inside double quotes without any escape character.

[S10:L116]

* Variables in JS doesn’t have a data type itself.
* U can reassign a variable of number type to e.g. string or any other, therefore JS is dynamically typed.
* Variables are either camelCase, snake\_case, kebab-case. JS variables must be camelCase.

[S10:L117]

* Uninitialized variables have a type “undefined”.
* Undefined means something that doesn’t have a value yet whereas null means explicitly nothing.
* Commenting in JS using //
* == doesn’t check the type e.g. 5 == “5” is true, null == undefined is true.
* === checks the type e.g. 5 == “5” is false, null === undefined is false.
* NaN means not a number.

[S11:L127]

* JS associates truthy and falsy value with every thing when evaluated in a boolean context.
* Falsy values: false, 0, “”, null, undefined, NaN.
* Truthy values: everything other than falsy above.
* Keep your code DRY (don’t repeat yourself).

[S12:L143]

* naming functions follow the same convention as variables i.e. camelCase.
* Every function returns something and if we dont have a return statement it returns undefined.
* In console of google chrome the return value would appear besides a little left-arrow.

[S12:L146]

* Two ways of defining functions:
* Function declaration: function fun{}
* function expression: var fun = function{}

[S12:L150]

* variables are passed by value i.e. a copy of variable is used in the functions.
* If you have 2 variables with the same name, one local and the other global, then inside the function the local variable would be used instead of global.
* Higher order functions are functions which takes functions as their parameters or which returns another function. E.g. setInterval()

[S13:L154]

* Two ways of initializing an empty string;
* Var friends = [];
* Var friends = new Array();
* Arrays don’t need to have the same type of data.
* forEach format: array.forEach(somfunction, index); e.g. to print the contents in ‘array’ you can use forEach as array.forEach(function(I, index){console.log(index “:” i)});
* NOTE: forEach works only for arrays so if something is not an array e.g. is a list then use the for loop.

[S13:L165]

* Just by mentioning the name of the function, the function wont run, you also need to put the brackets.
* Array.prototype contains all the methods for the array.
* Creating your own functions:
* Array.prototype.myForeach = function(func){func(this[i])};

[S14:L166]

* Objects property has a key-value pair.
* To create an object var obj = {key-value}
* To retrieve a value u can either:
* use dot notation e.g. obj.key;
* use bracket notation e.g. obj[“key”]
* if we have the property key with a space in it or starts with a number then dot notation would not be valid and hence bracket notation would be a better choice.
* Two ways to initialize an empty object;
* Var obj = {};
* Var obj = new Object();

[S14:L171]

* We can add methods as a property of an object and its called functions.
* Underscore is a popular javascript library.

[S15:L175]

* DOM (document object model) is an interface between the JS and HTML + CSS. Basically a bunch of JS methods that we can use to interact with out HTML and CSS. We can change things, animate elements. DOM is how we make our JS to make things interactive.
* DOM connects our JS with HTML and CSS.
* Its called document object model because all the objects are modeled inside the one object called the document(it’s the root node).
* Html turns into bunch of JS objects and each object models one of the many html elements.
* Browsers take each html element and turns it into an object.
* Console.dir(document) prints out the entire document object.

[S15:L176]

* Just like in CSS we selects elements we do the same in DOM but a little differently.
* Var varH1 = document.querySelector(“h1”), selects the h1 element from the HTML and save it in the varH1 variable. When just varH1 typed on console, it would return us the html but its actually a JS object to visualize this u can use console.dir(varH1).
* We select some elements from the html and we manipulate them.

[S15:L178]

* 5 main DoM methods:
* document.getElementById() : selects one element because ID’s are different. No need to mention “#”
* document.getElementsByClassName() : selects all elements with the mentioned class name and returns all in a node list. document.getElementsByClassName("special")[0] would give the first element with class “special”. No need to mention “.”
* document.getElementsByTagName() : selects all the element that match the tag name and returns all in a node list.
* document.querySelector() : selects and returns the first matching css-style selector. For classes we use ‘.’ And for ID’s we use ‘#’ e.g. document.querySelector(“#bomb”), selects the first element with ID = “bomb”. You can use all sorts of CSS selectors in the brackets. You cant index because it returns only one element.
* document.querySelectorAll() : selects and returns all matching css-style selector. For classes we use ‘.’ And for ID’s we use ‘#’ e.g. document.querySelectorAll(“#bomb”), selects all elements with ID = “bomb”. You can use all sorts of CSS selectors in the brackets. document.querySelectorAll(".special")[0] would give the first element with class “special”.
* we can replicate the above 3 with querySelector.
* JS should only control the behaviour of the structure.

[S15:L181]

* To manipulate the CSS properties use either:
* a class or
* calling the style object properties on it.
* We can define a class in CSS and add in our JS file.
* By adding classes to the JS we ensure sepration of concern by making sure that we are not doing any CSS related work in our JS file.
* classList is a list of classes assigned to a given elements. It is not an array.
* Var.classList.add(“myClass”), assign class “myClass” to variable Var.
* Var.classList.remove(“myClass”), remove class “myClass” from variable Var.
* Var.classList.toggle(“myClass”), assign class “myClass” to variable Var if not already assigned and if already assigned then it removes it.

[S15:L182]

* To manipulate the HTML TEXT use either:
* textContent.
* innerHTML.
* textContent returns everything contained in the tag as a string. Then u can update or play around with it.
* innerHTML also returns a string but unlike textContent, it also returns the tag contained inside the specified tag.
* Its dangerous to update using the textContent and innerHTML as this will update whatever is contained inside of it e.g. maybe a list or a image.
* Difference: innerHTML treats what is given to it as an HTML but the textContent treats it as a text e.g. document.querySelector(“h1”).textContent = “<h1>NEW Header</h1>” would change the h1 to “<h1>NEW Header</h1>” but document.querySelector(“h1”).innerHTML = “<h1>NEW Header</h1>” would change the h1 to “NEW Header”.

[S15:L183]

* To manipulate the HTML Attributes (not the tags) use either:
* getAttribute(“img”).
* setAttribute(“img”,”pic.png”).
* Remember p, span, div, body etc are the tags and not the properties. Get/setAttributes works with properties only e.g. width, height, src, href etc.

[S16:L186]

* To make some element interactive add event listener, e.g. button.addEventListener(“click”, function(){console.log(“u clicked the button!”);})
* U can add more than one listener to an element and they would be executed in order i.e. first listener first, second goes next and so on.
* You can either alter the CSS, HTML, attributes in the function of event listener.
* Types of eventListener:
* click: triggered only when the element is clicked.
* change: triggered when the element is changed either by clicking or through keying in data manually or through any other ways.
* mouseover: triggered when mouse is hovered over an element.
* mouseout: triggered when the mouse is moved away from the element e.g. to cancel the effect of the mouse hover.
* U don’t have to create an anonymous function inside the eventListener, u can also create a named function and just provide the name without the parenthesis in the eventListener.
* Var.value gets the values from the input field and return it as a string.
* Var.style.display = “none”, hides the Var from the view and Var.style.display = “block” displays it.
* CSS property “transition” would make the property fades out after the specified seconds. E.g. button{transition : color 1.0s;} would make the color property of button to fade for 1 sec. you can also give “all” instead of “color” to make transition for every single property of the button. The transition property is not built-in to every single browser so we have to also add “-webkit-transition: color 1.0s” and also “-moz-transition : color 1.0s” for more browser support.
* “border-radius” property of CSS changes the border style.
* CSS property “outline : none;” disable the outline around e.g. a button, when its selected.
* Ternary operator : name===”Fahad”? correct = true : correct false;

[S18: L204]

* jQuery is a DOM manipulation library (javaScript Library).
* It’s a single JS file with bunch of methods and features written inside of it and we can add it to our projects.
* Vanilla JS is JS without using jQuery.
* Almost anything u use jQuery for could be done without using jQuery.
* U can add jQuery using either CDN or download jQuery file.
* The compressed version is faster than the uncompressed.
* To check if the jQuery is installed properly, u can type “jQuery” on the console and if there are no errors then its working fine.

[S18: L208]

* For selection u use $ sign and for manipulation u use css function.
* $ sign is a function and returns a list of the elements, even if there is only one element.
* The $ sign in jQuery replaces the document.querySelectorAll() is DOM. E.g. $(“img”) would return all the image tags, $(“.title”) would return all the elements with class title and $(“#id”) would return the element with id “id”
* css method is jQuery, change the css property. css(property, value). You can also create an object with all the changes you need as the attribute in the object and pass it to the css().
* Unlike DOM, in jQuery, when we have a selection of elements, we don’t have to loop through each of them, jQuery automatically applies the property to each single one of them.
* We always include the library before we use them e.g. in head.

[S18: L210]

* “this” in vanilla JS wont work in jQuery we need to use “$(this)” instead. Its basically used to refer to the element in use.
* Any eventListener can be used with “on”. If we use “on” e.g. $(“button”).on(“click”, function(){alert(“button clicked!”)}) then all the potential future buttons will alert “button clicked!” when clicked but without “on” e.g. $(“button”).click(function(){alert(“button clicked!”)}) would only alert “button clicked” when the buttons existing at that time are clicked and not alert if any button in the future is clicked.

[S20: L223]

* Event bubbling: the events listener will be triggered outwards to any layers of listeners that are added.
* When we say this.parent() and then we say $(“this”) again then now we are working with the parent and not the child.
* $(“ul”).on(“click”, “li”, function(){ alert(“LI clicked”); }) is adding action listener to “li” and not to “ul”. It basically means that whenever in the future a new LI is created and we press on it then it should alert “LI clicked” this is a way to add event listener to the future elements based on what we have now.
* uigradients.com is a great resource to get the gradients for the background.

[S21: L231]

* we would be using paperJS library for graphics and howlerJS library for sounds in the patatap clone project.
* “canvas” is an HTML element that is used along with JS to draw graphics.

[S22: L240]

* DNS takes the domain name and convert it to IP address and an HTTP request (your IP address, time you are sending the request, type of request etc.) is send to that IP address. The requested server responds with the combination of html, css and js, and then the browser converts this data into something that we can see.
* Stack refers to all technologies(backend technologies (e.g. python, java, JS), server (e.g. nginx), database (e.g. MySql, postgreSQL) and front end (html, css, JS)) that a particular web app uses (html, css, JS are always part of the stack). stackshare.io lists different companies stack.
* Full stack web development refers to writing code for all parts of the stack i.e. client and server side.
* The stack that we would be using is NodeJS(backend), Express(server), MongoDB(database).
* Server-side code includes backend technologies, server, database and client-side code includes JS, HTML, CSS. To make dynamic web pages we need server side code.

[S22: L242]

* Postman app help us make HTTP request and view responses and debug if the response is not coming back the same way.
* We can make request from browser, command line, postman, backend and from many different places.
* Type of requests:

1. Get: for retrieving data e.g. asking for something from website server.
2. Post: sending some data along with request/ posting new information to a database e.g. submitting a post or commenting. We can make POST request using HTML form.
3. Put/patch: developer made this request to edit website after they have been posted.
4. Delete: when server gets delete request then it deletes that.

* There are 2 important part of every response from a server for a request

1. Body (HTML, CSS, JS), the one you see when you click “view page source”.
2. Headers, its contains the status code e.g. 404, 200

* “rm -rf folder” removes the entire directory

[S24:L254]

* Node allow us to write the JS code on the server side. Before node, other languages like pthon, java etc were used to write server side code but JS was not an option.
* The reason for using nodeJS is that with it we don’t have to learn another language to write the server side code.
* To exit from the node console, press ctrl C twice.

[S24:L258]

* Npm is the node package manager for the javaScript, it comes with Node and it lets us install packages. It’s a way of including libraries/packages to the node.
* To install a package, you use “npm install” command.
* To include the package, we use “require”.
* Packages are automatically installed inside “node\_modules” directory.

[S25:L263]

* Both framework and libraries are external code that you are including in your application but for library you have the control like jQuery but in framework we have some decision that have been made for us like express so we have less control in framework and it is used to make the application development faster.
* Some examples of framework.
* Flask for python.
* dJango for python.
* Rails for ruby.
* Express for node.
* Express is a lightweight node web development framework, it has a lot of blanks for the user to fill in.

[S25:L265]

* Routes are the code that are responsible for listening/receiving request and deciding what to send back.
* Every time you edit you app then you have to restart the server. To avoid this you can use “npm i -g nodemon --save” and it will automatically restart the server on every change.

[S25:L265]

* Package .json stands for (java script object notation), every node package has this .json file which contains meta information about that package.
* “npm init” creates a package.json file for the directory you are currently in. using “--save” at the end when installing packages automatically add that package along with the version to the package.json dependencies object.
* For undefined routes you will have a get request with “\*” at the end of all the routes.
* The order in which we define the route matters e.g. if we put the “\*” route on top then its gonna catch all the routes.

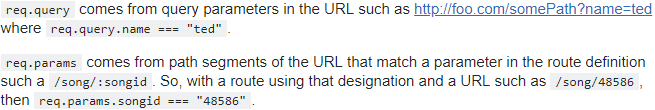
[S26:L272]

* You need to import the package “ejs” before you can use the file of this extension.
* “render” method of the “res” takes an ejs file(embedded JS) and parameters as objects, to send it as a response. We don’t usually send plain HTML file with render. This ejs file must be created inside the “views” directory because this is where the express would be looking for the file.
* “ejs” is a embedded javaScript which let us embed JS in our HTML. It enables us to have dynamic templates. To embed JS in ejs file you put the JS inside <%= %>.
* “ejs” has 3 kinds of tags, one with “=”, one with “-” and the other without equal sign. We use the “=” tag when we want to display something evaluated by that tag and we use the without equal tag when we just want to compute the logic without displaying what’s contained in the tag itself e.g. for loop and conditional statement uses the without equal tag. The tag with “-”, would take the code inside the <%- %> evaluate that and then display it rather than just displaying it as a string as in case of a <%= %>.
* With “app.use(express.static( \_\_dirname + "/public"));” express knows that “public” directory should be looked in for the static assets (e.g. images, JS, CSS), it saves us the time to write the path of the file.
* With “app.set("view engine", "ejs");” we can tell express ahead of time that we would be using “ejs” files and therefore we wont have to mention the extension “ejs” everytime with the file name. so instead of writing “repeat.ejs” we can just say “repeat”.

[S26:L274]

* herf=”app.js” looks for the app.js file in the current directory but href=”/app.js” looks for the app.js file in the root directory.
* To include another “ejs” template in some “ejs” template you use “<% include file\_name %>”

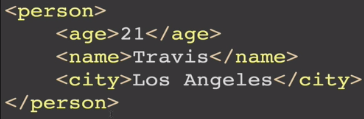
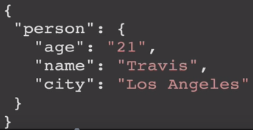
[S26:L276]

* npm package “body-parser” is used whenever we want to extract data from a form and use it in the server side. In order to use it, u need to require it first and then add “app.use(bodyParser.urlencoded({extended: true}));” where app = express();
* body-parser takes the req.body and parse it into JS object.
* 

[S27:L277]

* API is a way to interact with the code some other person wrote using the http request.
* ifttt.com, programmableweb.com are some websites which helps connect other apps API to your app.

[S27:L278]

* there are 2 formats that an API respond with JSON and XML, API doesn’t respond with HTML.
* With HTTP request we get the HTML back which contains information about the structure of the page. API’s respond with data and not structure.
* XML:
* Stands for extended markup language.
* Syntactically similar to HTML.
* 
* JSON:
* Stands for Java Script Object Notation.
* Looks exactly like JS objects but everything is a string.
* 

[S27:L279]

* You can also make an HTTP request from your terminal using “curl” command e.g. “curl [http://fahadq.com](http://fahadq.com/)”.
* To make API request using node we use a node package called “request”. Request can also take a call back function(error, response, body)
* Status code 200 is the standard response for a successful request.
* The data that we get after we made a request is a string therefore we need to parse it into a JS object, to convert a string into a JS object we use “JSON.parse(body)”.
* “req.query.search” would return the value when the request is made with the query name “search”

[S28:L293]

* Bootstrap class “form-control” takes up the entire width of the container it is in and also give the input fields the bootstrappy look.
* Bootstrap class “btn-block” also takes up the entire width of the container it is in but it should not be used with input fields, should be for buttons only.

[S29:L294]

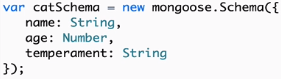
* There are 2 broad categories of databases: SQL and noSQL.
* Is noSql we don’t have any tables, things can be nested. Looks very similar to JS called BJSON(binary JS object notation), basically JS objects that have key-value pair.

[S29:L296]

* mongoDB is a noSQL database

[S29:L299]

* mongoose is an npm package, its a mongoDB object modeling for node.js
* Mongoose is not necessary but with mongoose its easier to interact with the database just like jQuery makes it easier to interact with the DOM.
* Before we can use mongoose:

1. we need to connect it to our DB. To connect we require it in our JS file and then use command “mongoose.connect("mongodb://localhost:27017/yelpCamp", {useNewUrlParser: true” and this will create a database called “catApp” if it doesn’t exists already but if it does then it will continue using the existing one.
2. We then need to define the schema of our database e.g. 
3. After that we need to define a collection. E.g. “var c = mongoose.model(“Cat”, catSchema);” which basically creates a collection “Cat” that would have the catScehma defined in step 2 and we are storing this in the variable ‘c’.

* We can use “find” to find something with a callback function. E.g. “Cat.find({}, function(err, cat){});” would return all the cats.
* We can use “create” to create an object and save it in the database. E.g. “Cat.create({name:”Meow”, age: 2}, function(err, cat){});”
* mongoDB automatically associates a unique ID “\_id” with every object it creates in the collection. We can search the DB with “findById” passing that ID to get that element info. E.g. “db.findById(req.params.id, function(err, objFound){});”
* to update the DB we can use “findByIdAndUpdate(id, returned\_data, function(err, objFound){})”, it takes 3 parameters.
* To delete an item by id we use “findByIdAndDelete(id, function(err, objFound){})”, it takes 2 parameters.

[S31:L305]

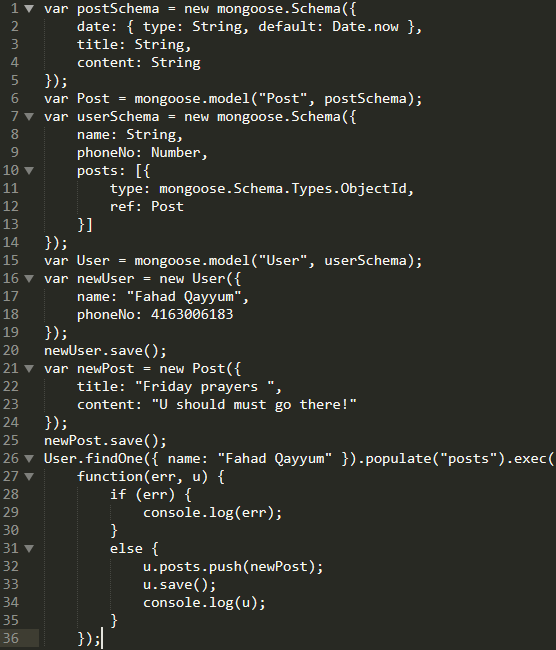
* we follow a convention for routing called REST.
* You cant send a “PUT” request using a form. Therefore if you make a “PUT” request it is treated as a “GET” request.
* The only requests that you can send using an HTML form is the “GET” and “POST”. Therefore, to send a “PUT” request in the form tag, under the action attribute, at the end you will specify “?\_method=PUT” and also you need to install package “method-override” which basically override the “POST” request to “PUT”. Then, you require the package in your file and add “app.use(methodOverride(“\_method”));”. Using this technique, you can change the POST request to any sort of request you wish.

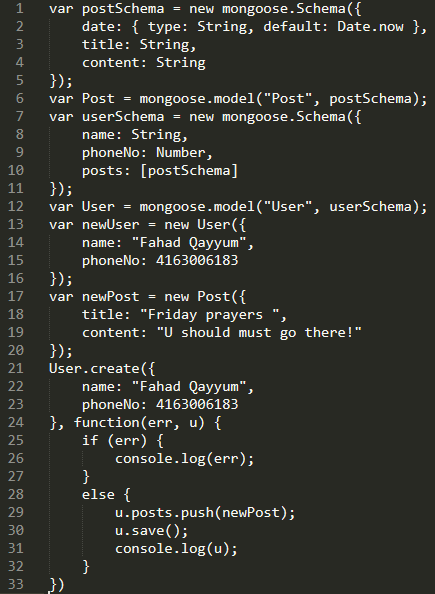
[S31:L316]

* To avoid user from giving malicious input you use the node package called “express-sanitizer” once you install it, then you require it in your program and after that you put “app.use(expressSanitizer());” after “app.use(bodyParser.urlencoded({extended : true}))”, if you are using body parser.

[S32:L321]

* Object references: in this kind of association, you first create an object of something and then you save it in its collection (not using create) and then you create the association and save it in its collection. Then you find the create object and then link it to its association. E.g.



* Without using line 3, the user posts would be just displayed with the IDs and not the actual content of the post.
* Embedding data is easier than object references. But in embedding you don’t save the newPost in its schema i.e. newPost.save(). But rather you create it like line 21 and then save it in the schema of the Users.
* 
* “module.exports” allows us to break things into file, this is important for making things reusable and cleaning up your code.
* module.exports = User;
* var postSchema = require("./post");

[S33:L324]

* to remove everything in a collection you use “collectionName.remove({}, function(err){})”

[S34:L334]

* passportJS is an authentication middleware for node.js. it has 500+ strategies to implement the user authentication.
* For authentication we need the following four packages:
* “passport”
* “passport local”
* “passport local mongoose”
* “express-session”
* Once you require “passport-local-mongoose” then you call a feature called “plugin” on your defined scheme e.g. “userSchema.plugin(passportLocalMongoose);” . with this our user schema gets a lot of important functionality and features for user authentication.
* Once you require “passport” you need to set it up by including the following 2 statements:
* app.use(passport.initalize());
* app.use(passport.session());
* once you install the “express-session” you can require and set it up like below:

app.use(require("express-session")({

secret : "idk",

resave : false,

saveUninitialized : false

}));

* template for setting up user authentication:

=================================Fill Out the Template============================

var express = require("express"),

app = express(),

mongoose = require("mongoose"),

passport = require("passport"),

localStrategy = require("passport-local"),

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

//put your schema here

var userSchema = new mongoose.Schema({

username : String,

password : String

})

userSchema.plugin(passportLocalMongoose);

var User = mongoose.model("User",userSchema);

app.use(require("express-session")({

secret : "idk",

resave : false,

saveUninitialized : false

}));

app.use(passport.initialize());

app.use(passport.session());

passport.use(new localStrategy(User.authenticate()));

passport.serializeUser(User.serializeUser());

passport.deserializeUser(User.deserializeUser());

==================================================================================

* “User.register(new User {username : req.body.username}, req.body.password, function(req,res){})” . here the register method on the collection “User” is taking 3 parameters. The 1st parameter is the user object that we wanna create, 2nd is password which would be hashed by the register method, 3rd is the call back function. So this will create a user with the specified username and the hashed password and another property called “salt” which help to unhash the hashed password.
* Whatever we put in “res.locals” is whats available inside of our template. The following is the way to pass “user” as a parameter to all the templates without explicitly mentioning this in the parameters of every render.

app.use((req,res,next)=>{

res.locals.user = req.user;

next();

})

* **NPM Packages:**

1. “ejs”
2. “request”
3. “nodemon”
4. “express”
5. “express-session”
6. “express-sanitizer”
7. “method-override”
8. “body-parser”
9. “mongoose”
10. “passport”
11. “passport-local”
12. “passport-local-mongoose”

* MongoDB Commands:
* “./mongod”: starts the mongo demon. We have to keep it running in order to use mongo.
* “mongo”: opens up the mongo shell. You can type mongo commands in there e.g. “help” , “show dbs”
* “help”: to get help with something.
* “show dbs”: list all the databases.
* “use” : create a db if doesn’t exist or if exists than switch to that db. E.g. “use newDB” create newDB if it doesn’t exists.
* “show collections” shows all collection of the current DB.
* “insert”: you can insert in to a collection in the current db by using insert. E.g. “db.dogs.insert({name:”rusty”,age:”3”})”, where db is the current database we are using, dogs is the collection and we are inserting in that collection a dog with name rusty and age 3. Mongo also automatically assign an “ObjectId” for every object in the collection.
* “show collections” shows the collection that are in the current db.
* “find” finds for a certain thing in the db e.g. “db.dogs.find()” returns everything in the collection dogs. Also, “db.dogs.find({breed:”Mutt”})” retruns all the dogs in the dog collection which has breed “Mutt”.
* “update” updates the value of an object in the collection. E.g. “db.dogs.update({name: “rusty”}, {$set: {name : “Lulu”}})” would update the name of “rusty” in the dogs collection to “Lulu” while keeping the other attributes same, If “$set” is not used then every attribute of “rusty” would be removed and only one attribute remains i.e. name which would be “Lulu”.
* “remove” something from a collection. E.g. “db.dogs.remove({name: “rusty”})” would remove all the dogs with name “rusty”.
* “drop” drops the collection in the database e.g. “db.cats.drop()” will drop all the collection of cats of db.
* CSS properties:
* box-shadow: 0 0 3px rgba(0,0,0,1); add a shadow around the element of 3px black with opacity 1.
* li:nth-of-type(2n){background-color : white} makes every 2nd li background-color white.
* Line-height property lets the line take up the specified space.
* box-sizing: border-box
* input:focus{background-color : white;} makes the background-color to white when we are working with the input.
* To encode a space in the URL u use %20.
* jQuery Functions:
* text() features returns the text like textContent in vanilla JS. Whatever text is given within the parenthesis is treated as a string.
* html() features is just the same as innerHtml is vanilla JS. If html is given between the parenthesis then its treated as html.
* attr() features can either be used to get the property value or set the property value.
* val() features is used to retrieve or set the value of the input. E.g. $(“input”).val(“Hello”) would set the text input to “Hello”.
* To stop event bubbling to propagate triggering we can use feature called stopPropagation(). It basically stops from triggering other possible event listeners.
* $(this).parent() gives us the parent of the current element.
* append() adds something at the end to whatever we selected using $ sign. E.g. $(“ul”).append(“<li>This is a new item<li>”) would add “This is a new item” at the end.
* jQuery EventListeners:
* click() takes a function and add the click function listener. E.g. $(“.btn”).click(function(){alert(“button clicked”)}).
* keydown() takes a function. It is triggered as soon as the key is pressed down.
* keyup() takes a function. It is triggered as soon as the key pressed is released.
* keypress() takes a function. It is triggered when a key is pressed. It has a property called “which” that returns the code for the key pressed. E.g. $(“input[type=”text”]”).keypress(function(event){if(event.which === 13){alert(“you hit enter!”)}})
* on() works similar to addEventListener, it lets the user specify the type of event to listen for. E.g. click or dblclick or dragstart or keypress etc. all the eventListener can be used with “on”
* “mouseenter” is the same as “mouseover” and “mouseleave” is the same as “mouseout” in vanilla JS.
* jQuery Effects:
* fadeOut() takes an element from whatever opacity its currently at and fade it to full transparency. E.g. $(“h1”).fadeOut(1000, function(){alert(“Fade out completed!”)}) would fade out the h1 taking 1sec and after that one sec it would alert “Fade out completed”. fadeOut only hides the element and to remove the element we can use $(“h1”).remove() in the function.
* fadeIn() fades all the element in.
* fadeToggle fade in or out depending on what state it is in currently.
* slideDown() show element sliding down from display=”none” and then it set display = “block”. E.g. $(“h1”).slideDown(1000, function(){alert(“slided down”)}).
* slideUp() shows the displayed element sliding up and then set the display to none.
* slideToggle() toggle the slideUp and slideDown.

vanilla JS functions list:

1. function str.replace(/-/,”\_”) replaces all ‘-’ with ‘\_’.
2. Functions toUpperCase() capitalizes e.g. “string”.toUpperCase() returns “STRING” whereas “string”.chatAt(0).toUpperCase() returms “S”.
3. Function slice(), slice the string and return the remaining portion e.g. “string”.slice(4) would remove the first 4 chars and return remaining i.e. “ng”.
4. indexOf() returns the index of the string or the character in the string e.g. if str = “Fahad Qayyum” then str.indexOf(“Q”) returns 6 whereas str.indexOf(“Qayyum”) also retruns 6. If the character or the word is not in the string e.g. ‘z’ then it returns -1.
5. Typeof function tells the type of the variable.
6. To typecast a var into a number we do Number(var).
7. Alert(), whatever is put in the brackets is prompted on the screen weither u put string, number etc.
8. Prompt(), is also like alert but with prompt we can ask user from input. Prompt takes the user input as a string, so using === to compare with a number wont work and we have to typecast.
9. We can use .length property on strings to get its length. It would also count the spaces. E.g. “H I:12”.length would return 6. NOTE: “hi\\”.length is not 4, its 3 because one is just an escape character.
10. Function setInterval(functionName, time) takes 2 parameter, one is the name of function and the other the time, this calls the given function after every time unit specified in the parameter. It returns a number which is passed to clearInterval() to stop it. It’s a higher order function.
11. Function setTimeout(functionName, time) takes 2 parameter, one is the name of function and the other the time, this executes the function after waiting the specified number of milliseconds.
12. Math.random() \* 5 generates numbers between 0-4 inclusive.
13. Array functions:
    1. Push() adds element to the end of the array and returns the new length of the array.
    2. Pop() removes the last element from the array and returns it.
    3. Unshift() adds something to the beginning of the array and returns the new length of the array.
    4. Shift() removes the first item from the array and return that item.
    5. indexOf() takes an item to find within the array and return its index, if multiple item found then it returns the index of the first one, if no item match the parameters the it returns -1.
    6. Slice(a,b) copies different portion of the array from index ‘a’ inclusive to ‘b’ excluded. Slice(a,a) would have no element to copy as it means include ‘a’ and exclude ‘a’.
    7. Splice(a,b) where ‘a’ is the index to delete and ‘b’ is the number of items to delete following that index. E.g. let array = [1,2,3,4,5] and array.splice(2,3) would delete 3,4,5.