ES6 and Fundamentals of Web Development

CS-554 – WEB PROGRAMMING

Course Technologies

Our Tech Stack

By default, we will be starting with the Node, Express, Mongo tech stack

- Direct continuation of Web 1 web stack, with tons more!

Most instructions will assume a Mac OS X operating system in this course.

Recommended Operating Systems

Windows is **not** a recommended operating system for this course, and Mac OSX will be the assumed OS in all instructions due to it being the ideal web development environment.

If you do **not** have a Mac system, you can emulate it using VirtualBox

- https://techsviewer.com/how-to-install-mac-os-x-el-capitan-on-pc-on-virtualbox/

Node Version

In this course, we will be using Node for our server side language. You should install the most recent **current** version of Node for this course.

- https://nodejs.org/en/download/current/

You may find it easier to install with NVM to maintain many versions of node

- https://github.com/creationix/nvm
- https://github.com/coreybutler/nvm-windows

Mongo DB

For most scenarios, we will be using MongoDB for the de facto database of choice.

- https://www.mongodb.com/download-center#community

It is an easy, simple, key-value database.

Redis

Later on in the course, we will be using Redis.

- https://redis.io/

I implore you to install this now and early and experiment with it.

ES6

CHANGES TO THE JAVASCRIPT LANGUAGE

What is ES6?

ES6 is the most recent spec of what constitutes the JavaScript language. It is, at this point, mostly adopted on the Node side.

ES6 provides a series of **very** large conveniences, especially with easier asynchronous code handling.

What is async / await?

Handling asynchronous code often leads to massive lines of confusing promises, or fiendishly long callback chains.

In order to eliminate this, ES6 introduces a syntax for unraveling a promise, known as *async* and *await*.

This is, single handedly, the best thing to happen to the language since its creation.

What does async mean?

You can now define a function as an “async” function. This means that the function body will automatically be wrapped in a promise, and will return a promise (even if, syntactically, it just returns a value without any async code).

Async functions have the benefit of being able to *await* promises (and, therefore, other async functions).

Async functions are the only types of function that can *await* promises.

What does awaiting do?

Using the *await* keyword will, syntactically, not execute the next line of code until the promise has been resolved or fulfilled. This is syntactic sugar only. In actuality, the code below that await is unraveled and placed in a *.then* callback for the awaited promise.

As a result, you can now write async code that appears to be written synchronously.

As a secondary result, you can now simply use try / catch around your async code and catch errors that way.

Example

Remember, we can only await inside async functions.

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

```
function resolveAfter2Seconds(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x);
    }, 2000);
  });
}

async function add1(x) {
  var a = resolveAfter2Seconds(20);
  var b = resolveAfter2Seconds(30);
  return x + await a + await b;
}

add1(10).then(v => {
  console.log(v);  // prints 60 after 2 seconds.
});

async function add2(x) {
  var a = await resolveAfter2Seconds(20);
  var b = await resolveAfter2Seconds(30);
  return x + a + b;
}

add2(10).then(v => {
  console.log(v);  // prints 60 after 4 seconds.
});
```

Default Parameters

Now, we can set default values to our parameters. If a parameter is *undefined*, it will take on the default value.

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters

```
1  function multiply(a, b = 1) {
2    return a * b;
3  }
4
5  multiply(5, 2); // 10
6  multiply(5, 1); // 5
7  multiply(5);    // 5
```

Template Literals / String Interpolation

```
1  console.log(`string text line 1
2  string text line 2`);
3  // "string text line 1
4  // string text line 2"
```

```
1  var a = 5;
2  var b = 10;
3  console.log(`Fifteen is ${a + b} and
4  not ${2 * a + b}.`);
5  // "Fifteen is 15 and
6  // not 20."
```

Rather than concatenating strings like barbarians, we can now just interpolate our strings.

We can also make multine strings in the same format. Finally.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

Object Literal Shorthand

Gone are the days of defining an object that uses the same key name as variable names, and writing it twice!

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer

```
1 // Shorthand property names (ES2015)
2 var a = 'foo', b = 42, c = {};
3 var o = {a, b, c};
4
```

Destructuring Objects

Conversely, we can now go from ”object with these 5 properties” to “these 5 variables”

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

Basic assignment

```
1 | var o = {p: 42, q: true};
2 | var {p, q} = o;
3 |
4 | console.log(p); // 42
5 | console.log(q); // true
```

Destructuring Arrays

Similarly, arrays work the same way.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

Basic variable assignment

```
1 | var foo = ['one', 'two', 'three'];
2 |
3 | var [one, two, three] = foo;
4 | console.log(one); // "one"
5 | console.log(two); // "two"
6 | console.log(three); // "three"
```

```
let {a, b, ...rest} = {a: 10, b: 20, c: 30, d: 40}
a; // 10
b; // 20
rest; // { c: 30, d: 40 }
```

```
var obj1 = { foo: 'bar', x: 42 };
var obj2 = { foo: 'baz', y: 13 };

var clonedObj = { ...obj1 };
// Object { foo: "bar", x: 42 }

var mergedObj = { ...obj1, ...obj2 };
// Object { foo: "baz", x: 42, y: 13 }
```

Object Spread

While destructuring, we can also spread an object and create another object with those keys.

This also makes for good shallow cloning, as well. It is not a deep clone operation. You can shallow-clone an object by using the spread operator without other parameters, or you can override

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator

Web Development Recap

Intro to JavaScript and Node

LECTURE 1 – WEB 1

Some basic facts about JavaScript

JavaScript is a loosely typed language, a concept that you may have seen before.

- Loose typing means that you don't strictly declare types for variables, and you can change the type of data that that you store in each variable.

There are five primitives currently, with a sixth (Symbol) on the way

- Boolean
- Number
- String
- Null
- Undefined

JavaScript also has Objects, which all non-primitives fall under

- Functions in JavaScript are types of Objects
- Objects are prototypical

Functional Scope in JavaScript

Functions are one of the most fundamental building blocks of JavaScript.

We often want to isolate our scope in JavaScript, particularly when we write browser-based JavaScript code in order to avoid conflicts between libraries.

While Node.js scripts isolate their variables between files, all top-level variables in a browser-environment become global variables, even across different files.

In JavaScript, scope is not defined by block unless using the keyword *let* to define a variable; when using *var*, it is defined by the function you are in.

- http://coffeegrammer.com/understanding-functional-scope/

JavaScript resources

Most of the JavaScript language can be learned from two simple guides; the MDN and the Node manual.

The Node Manual contains all information on JavaScript in the Node environment

- https://nodejs.org/api/

General and front-end based JavaScript can be found on the MDN

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

MongoDB

LECTURE 4

What is a database?

A database is an organized collection of data. It allows you to create, read, update, and delete data. Unlike storing in memory, databases allow you to persist your data.

MongoDB is a **document-based database**.

- **Document-based** databases store semi-structured data (think, JSON!) and only lookup by key (a unique identifier)

You interact with MongoDB by submitting queries to your database that describe operations that you wish to do.

What is a document-based database?

Traditional databases are stored in tables that are composed of columns describing data and rows of data. They have a pre-defined schema to them, constraining the type of data you can make in each table.

Document-based databases forgo this, and allow you to simply store and retrieve data at a particular location. They are **very** good at ID lookups, but suffer slightly on querying.

- This is less off an issue now than in previous years.

The structure of MongoDB

MongoDB only has a few layers to it:

1. Databases: You can create a database to contain related collections
2. Collections: Each database has a number of collections. Collections are sets of documents that you **decide** are related by their content. Your documents do not have to have the same fields.
3. Documents: Documents are self contained pieces of data that you store in a collection. Each document must have an ID, and can have any other set of fields; these fields can be smaller subdocuments.
4. Subdocuments: A document field can describe another document that will be stored in its parent. This is akin to an object that has a second object stored as a property. This is referred to as a subdocument.

Abstracting Your Queries

It is often **very** useful to create a file to abstract away your database querying.

By creating a layer between your application code and your database, you will allow yourself to:

- More easily make changes to the database later on
- Allow non-database programmers to more easily use the database (separation of concerns!)
- More easily improve performance at a later time
- **Easier and more consistent error checking throughout your entire application**.
- Make it a reasonable task to change your entire database when the first database your company chose ends up being unable to support large amounts of data and you need to transition over to another database.
  - Also helps when this happens again 2 years later.

In **dogs.js** we abstract the queries to hide them away from the rest of our application.

Basic operations in MongoDB

For now, we will be focusing on four different operations:

- Insert: will take an object and insert it into the database.
- Find: will take an object describing fields and values to match and returns an array of matching documents.
- Update: will take two objects; one that contains an object describing fields and values to match, and one that will describe the update to perform. It can update multiple if you provide a third object with settings telling it to update multiple documents.
- Remove: will take an object describing fields and values to match and will remove the object. It can remove multiple if you provide a second object with settings telling it to update multiple documents.

MongoDB Cheat Sheet

Most common MongoDB commands can be found on this cheat sheet

- https://blog.codecentric.de/files/2012/12/MongoDB-CheatSheet-v1_0.pdf

Most MongoDB commands in the Node Driver have fairly straightforward commands.

- http://mongodb.github.io/node-mongodb-native/2.2/api/

Fundamentals of Web Development

LECTURE 5

The Core Process of the Web

At the end of the day, the web is all about the communication of ideas. Everything in the web can be seen as a **request** and a **response**

- When you go to a news website, you're requesting news and receiving news in response.
- When you go to a shopping website, you're requesting product information and receiving relevant information.
- When your server receives input, it determines what to do with that input and outputs the proper response.

Your duty as a web developer is to make that communication possible. Your programs will get a request and give a response, and allow that communication to occur as smoothly as possible.

The request

Every time you navigate to a website, your browser sends a **request** on your behalf to the server.

- There is a lot to an HTTP request! https://developer.mozilla.org/en-US/docs/Web/HTTP
- Each request follows a standardized process! https://www.w3.org/wiki/How_does_the_Internet_work

Every request is formatted in a specific way, with the same data provided on each request.

A request to http://google.com/ would look like:

```
GET /?gws_rd=ssl HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Cookie: PREF=ID=1111111111111111:FF=0:TM=1441084937:LM=1441084937:V=1:S=DKD8wPI-NAGQztx5; NID=71=0zCx
Connection: keep-alive
Cache-Control: max-age=0
```

Parts of that request

HTTP Verb signifying what action you are trying to (GET POST PUT DELETE)

The server you want to connect to (the HOST)

The location of the resource you want to access on that server (the location)

Headers

- Headers are metadata about your request
- These include cookies!

Status Codes

Each response must return a status code indicating whether or not the request was successful, and a description is often included with each of the status code

Status codes in the...

- 200-299 range indicate a successful operation
- 300-399 range indicate some sort of redirection must occur
- 400-499 range indicate an error was made by the client during the request
- 500-599 range indicate some sort of error occurred on the server

You will use different status codes to describe different errors in this course

Some status codes:

- https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Express

Express is a very popular node package that is distributed on NPM which allows you to configure and run an entire web server.

- http://expressjs.com/

Express allows you to configure different routes and how they should compose a response.

Essentially, by using the Express module, you will use code to configure a server that will listen to requests and send out responses.

- **This is all a web server is! It's not magic, it's just something that takes in requests and sends back responses!**

Fundamentals of HTML and CSS

LECTURE 7

What's in an HTML Document?

An HTML has a series of elements
- Open tag plus attributes and properties
- Nested elements

Some very important
- HTML Doctype
- HTML Element
  - Head Element
  - Body Element

Elements can be identified by an ID
- ID can only be used once per document

Elements can identify a group by their class

Elements are described in the document and rendered in the DOM

The Browser's Only Half The Battle

The web isn't just accessible via a browser. As modern web developers, we have to care about:

- Screen readers
- Search Engine Crawlers / Other AI

There is a growing movement to make the web more accessible

- Leveraging HTML's strengths
  - Navs in the nav
  - Labels in forms
  - Using headings properly
  - Attributes to help screen readers
- Making designs accessible
- Tables for tabular data only
- Make sure you can navigate via keyboard

Separating Style and Content

Before we can think in terms of organizing our data meaningfully, we need to understand what HTML does *not* accomplish; the way your document looks.

- **Elements are used to describe your data; CSS is used to style your data.**
- While browsers give many native styles to elements by default, elements are not inherently used for styling. This is why tags for bolding and italicizing text, or changing fonts, were deprecated in HTML5.
- There needs to be a clear separation between style and content; any overlap is a happy coincidence.

While writing HTML, thinking in terms of content first, then styling often leads to more logical, and easier to style documents.

Types of Text

Across the web, text is used to portray many different types of things.

- Headings / Titles in your content (h1, h2, h3, h4, h5, h6)
- Regular paragraph (p)
- Generic groups of text / adding custom definitions or functionality to text (span)
- Emphasized text (em)
- Important text (strong)
- Addresses (addr)
- Citations (cite)
- Abbreviations (abbr)
- Quotes (blockquote)

Using the right kind of element to describe text is very important for SEO, non-browser accessibility, and readable code.

- Even without different styles, those tags help readers understand their document.

The layout of your content

There are many elements that describe the layout of your content

- How to navigate content / your document (nav)
- Grouping your content into sections that have something to do with each other (main, section)
- Denoting a header for content or your document (header)
- Denoting a footer for content or your document (footer)
- Grouping content into a self-contained article (article)
- Stating that certain content is secondary (aside)
- Grouping divisions of content (div)

Validating HTML

For this course, the validity of your HTML is highly important.

Having valid HTML means your browser does not have to guess how to fix it, which can lead to drastically wrong web pages and pages that cannot be made sense of.

The w3 website has an easy to use validation service that tells you issues and proposed solutions:

- https://validator.w3.org/#validate_by_input

You should view the source of your page, copy, and paste it all into the HTML validator's 'direct input' section before submitting HTML in this class.

You should strive to write as perfect HTML as possible.

Attributes and properties

Elements can have many classes and properties attached to them to further describe them.

The difference between the two of those are nuanced and deals with the state of the page.

- This is an example of how browsers had to adapt to a set of standards that were not always fully thought out.

Attributes appear in key-value fashion when writing HTML:

- Go To Google

The *href* attribute is set to Google's home page

Elements are parsed and, as they are changed, keep track of the set of properties. Some properties come from their attributes, others come from user input.

Classes and IDs

Elements will often be described with classes and IDs to signify them in some way

Many elements can share a *class*, and each element can have many *classes*

-
  - Has two classes, panel and panel-default
-
  - Has two classes, panel and panel-danger

However, only one element can have a particular ID:

-

Classes and IDs are most often used to style elements and target elements with JavaScript to add functionality.

What is CSS?

CSS is the language that we **style** HTML documents with.

- Cascading
- Style
- Sheets

CSS allows you to define rule-sets, which are selectors (identifiers that target HTML Elements) and rules (rules that define visual properties)

You can use CSS to describe how a document is presented in different contexts

- In a browser
- On a projector
- When printed

It allows you to take your documents and make them look like fully designed pages.

Adding CSS

It is very easy to add a CSS stylesheet to your HTML document.

In the **head** element, you add a *link* element like such:

- `<link rel="stylesheet" type="text/css" href=”/path/to/style.css">`

How does CSS work?

CSS is a simple language based on only three pieces of data:

- A selector, which is a pattern that will match elements
- A declaration, which has:
  - A **property**, which defines what property you will update
  - A **value**, which defines what you want to do to that property.

A set of selectors and a set of declarations makes a *rule-set*

Each rule-set can have multiple selectors, and multiple declarations.

CSS Syntax / Example rule-sets

```
h2 {
  text-align: center;
}

p.bio, .about-me .career-info {
  font-size: 16pt;
  border-bottom: 1px solid #333;
}
```

The first rule-set will target **all** h2 elements in the document, and center all the text contained inside the element.

The second rule-set will target all **p** tags that have a class of **bio**, as well as all elements with a class of **career-info** that are contained inside an element that has a class of **about-me**; there can be any number of elements and nested layers of elements between **about-me** and **career-info**. Any matching elements will have a bottom-border that is 1 pixel in size, grey colored, and a solid line; they will also have their font set to be 16pt.

Selectors

CSS	Name	Selects
`*`	Universal	Any and every element
`div`	Element	All div elements
`.foo { }`	Class	All elements with a class of *foo*
`#bar { }`	ID	The single element with the id of *bar*
`#bar .foo { }`	Descendant	All *.foo* that are contained inside *#bar*
`.parent > .child`	Child	All .child directly inside of .parent
`.post + .divider`	Adjacent Sibling	All .divider that are directly after .post in their parent.
`.post ~ .subtext`	General Sibling	All .subtext that come anywhere after .post in their parent.
`.post:pseudoclass`	Pseudo-Class	All .post that have a particular pseudoclass.
`[role='navigation']`	Attribute	All elements that have an attribute named 'role' with the value of 'navigation'. You can do this for any attribute and value.

Pseudo Classes (All Types)

Pseudo Class	Selects
`:first-child`	The first child of a parent
`:last-child`	The last child of a parent
`:first-of-type`	The first element of a type, inside of a parent; does not accept a class or id.
`:last-of-type`	The last element of a type, inside of a parent; does not accept a class or id.
`:nth-child(an+b)`	Selects all .child directly inside of .parent The formula (*an+b*) describes which elements are targeted; elements start at index 0. (*2n+1*) would select elements at index 1, 3, 5, 7 while (*3n*) matches at index 0, 3, etc.
`:nth-last-child`	Same as :nth-child, except starts from the last element
`:nth-of-type`	As as :nth-child, except works with element types
`:empty`	An element that has no content, including whitespace; can have comments.
`:target`	The element matching the target of your current hash.

Size Units

Unit	Name	Description
`px`	Pixel	Size of a pixel on the screen
`mm`	Millimeter	Size of a millimeter
`cm`	Centimeter	Size of a centimeter
`in`	Inch	Size of 1 Inch; generally, 96 pixels per inch
`pt`	Point	Size of 1/72 inch
`pc`	Pica	Size of 12 Points
`em`	em	Calculates the size based on the elements font size, or parent's font size. Literally, relative to the "M" in a font at a size.
`%`	Percent	Calculates the percentage of a size relative to a property on the parent element.
`rem`	Rem’s	Same as em, but based on the root element. Allows for very responsive layouts just by updating the root element.

Color Units

Color	Unit Name	Description
`rgb(255, 255, 255)`	RGB	Allows you to describe colors as how red, green, and blue they are from 0 to 255 each. Allows for 16,581,375 colors.
`rgba(0, 192, 45, .5)`	RGB Alpha	Same as hex, where the last decimal is transparency from 0 (clear) to 1.0 (solid)
`#A90BEF`	Hex	A hex triplet; has 2 hex digits representing each color (Red, Green, Blue). Describes your color from Allows for 16,777,216
`hsl(0, 20%, 50%)`	Hue, Saturation, Lightness	Takes the hue from 0 to 360; 0/360 are red, 120 green, 240 blue. Saturation determines what percent of that color you use. Lightness is a mask over saturation, adding a white layer. 50% is the normal value.
`hsla(0, 20%, 50%, .5)`	Hue, Saturation, Lightness, Alpha	Same as above, where the last value is transparency.

Text Rules

Rule Name	Example	Outcome for selected element
font-family	`font-family: "Open Sans", "Helvetica", sans-serif;`	If the user has Open Sans font, will use open sans; else Helvetica; else sans-serif (generic font).
font-size	`font-size: 18pt;`	Sets font size at 18pt (about ¼ of an inch)
line-height	`line-height: 3;`	Will make each line have a height of 3x font size.
text-decoration	`text-decoration: underline;`	Text will be underlined.
font-weight	`font-weight: 700;`	Font will be bold; norm is 300.
text-align	`text-align: center;`	Text will be centered inside parent element.
font-variant	`font-variant: small-caps;`	Makes an element use small capitalized letters for lowercase.
font-style	`font-style: italic;`	Makes text italicized.
font	`font: 2em "Open Sans", sans-serif;`	Sets font at 2em large, Open Sans (fallback to sans-serif)

Color and Background

Rule Name	Example	Outcome for selected element
color	`color: #690;`	Changes text color to a green.
background-color	`background-color: #000;`	Sets background color to black.
background-image	`background-image: url(gradient.png);`	Uses gradient.png as background for element.
background-attachment	`background-attachment: fixed;`	Makes the background image fixed in browser.
background-clip	`background-clip: border-box;`	Sets the background to fill entire box model.
background-position	`background-position: left center;`	Positions your background anchored to the left horizontally and center vertically.
background-repeat	`background-repeat: repeat-x;`	Sets background to only repeat horizontally.
background-size	`background-size: contain;`	Scales the background image to fit length or width and letterboxes.
background	`url(gradient.png) no-repeat;`	Shorthand for multiple properties.

Setting a border

Example	Outcome for selected element
`border-left: 1px solid #999;`	Sets a 1px wide solid border on the left; grey.
`border-right: 5px dashed #558abb;`	Sets a 5px wide dashed border on the right; light blue.
`border-top: 1px solid #999;`	Sets a 1px wide solid border on the top of the element; grey.
`border-bottom: 1px solid #999;`	Sets a 1px wide solid border on the bottom of the element; grey.
`border: 2px dotted #558abb;`	Sets a 2px wide dotted border on all sides; light blue.

Putting it all together.

Example	Outcome for selected element
`box-size: border-box;`	Makes your element include padding and the border in width.
`padding: 5px;`	Adds 5 pixels of padding on the element.
`margin: 0px auto;`	No margins on top or bottom; left and right will automatically be calculated, possible centering element in parent.
`width: 75%;`	Sets width of element to be that of 75% of the parent's width.
`height: auto;`	Height will automatically be determined by content;
`max-width: 500px;`	Element will not exceed 500px wide.
`position: relative;`	Element will be moved relative to static position.
`top: 10px;`	Element will be 10px below where it was originally meant to be

Basic Web Accessibility

LECTURE 10

What is accessibility?

As web developers, we have the opportunity to make sure that our websites and web applications are consumable by people with a number of disabilities.

Even simple pages can have a number of issues that cause a person with some form of disability to be unable to fully use it; a form without labels, for example, is much harder for a screen reader to parse. A visually impaired user would struggle.

Even your design affects web accessibility: a lack of color contrast can make text nearly invisible to some of your users!

Testing accessibility

There are many ways we can test for accessibility, but to start, we will be using the tota11y tool

- http://khan.github.io/tota11y/

The tota11y tool is an accessibility visualizer that can be installed via a bookmarklet.

This tool will allow you to identify how assistive technologies would interpret your website.

Going forward

Going forward, **all HTML submitted must pass tota11y tests.**

Points will be deducted for labs and final project components that fail accessibility checks.

AJAX and Security

LECTURE 10

What is AJAX?

AJAX (asynchronous JavaScript and XML) is a series of techniques used to have clients execute JavaScript code in order to request resources without leaving their current page.

This means that you can write code that makes network requests on the client's behalf to access data on your server.

We will be using jQuery to perform our AJAX requests.

What does asynchronous mean?

AJAX calls are inherently asynchronous; you do not know when they will complete, so they pass data through callbacks.

- The request is made on a background thread that does not block the UI
- This request can finish anytime after it is sent, so you want to hook into your event listener before the request is sent; a request could theoretically complete before you listen for the response.
- The request may not ever complete successfully

Requesting Data (JSON)

With Express, having a route return JSON is very easy.

You can easily request and use JSON data by:

- Make an AJAX request to a route that sends JSON in its response
- Listen for a callback state change
- Check the data to see what you should do (ie, was it successful? Then render; otherwise, show error)
- Manipulate and use the data as needed

Submitting JSON

We can post JSON, as well! This is particularly useful, as you can finally start sending numbers and booleans as well as strings by sending JSON.

In order to POST data in JSON format, you format your request and pass JSON in the body of the request.

Using the bodyparser middleware allows you to have this JSON easily parsed into our request body field in our Express routes and middlewares.

Benefits of using AJAX

Smaller payloads; you can only send down data that the user cares about.

You can split up your code into a more modular setup.

You can keep updating one page, rather than re-requesting entirely new pages all the time!

- Since one page is updating, you will not have to re-request and re-render the same resources such as stylesheets and JS files, making your application often perform much better.

Middlewares

LECTURE 11

What is middleware?

A middleware is a function that has access to the request and response objects

These functions can:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

You can apply middleware to the entire application, or portions of the application

- You can apply it to a portion of the application by supplying a path as the first parameter to the middleware function

Practical Uses for Middleware

Middlewares are useful for a number of reasons, and have many common uses

- Logging requests
- Authentication
- Access control
- Caching data
- Serialization

Writing a middleware

Writing a middleware is extremely easy

- Register your middleware, optionally providing a path to apply that middleware to
- Have your middleware perform a task and when done:
  - Have your middleware end the response
  - Have your middleware call the next middleware

As an example, see the *server.js* file, which has several middleware:

- One which will count the number of requests made to your website
- One which will count the number of requests that have been made to the current path
- One which will log the last time the user has made a request, and store it in a cookie.
- One which will deny all users access to the */admin* path.

What is authentication?

Authentication is the act of confirming the identity of a person, group, or entity.

In web technology, this often means creating a **user login system**

- You will use a combination of data in order to identify a user.

There are many other forms of authentication in web technology:

- You can make an authentication system that allows you to limit API Access
  - Force users to have a token
  - Allow users a certain number of access hits a month
- You can selectively allow or dis-allow access to resources based on user login state

Implementing Authentication

In order to implement authentication and create a user login system, we will be breaking down the task into several steps:

- Creating and storing users
- Allowing users to login via a form
- Storing session data in a cookie
- Validating the data stored in the cookie
- Storing the user as part of the request object.

Let's walk through this process.

Logging Out

Logging out is extremely easy, and only has two steps!

- After hitting a logout route, you will expire the cookie for the session id
- You will remove the session id from the user's session id list
- You will invalidate any other cookies that are relevant to the user.

By doing both of those, you will have successfully invalidated the session and the user will no longer be authenticated.