Web Development Documentation

Release Spring 2019

Paul Vincent Craven

Jan 16, 2020

Contents

CHAPTER 1

Example Report - New Features in JavaScript ES6

ECMAScript 6, also known as “Harmony” and often shortened to ES6, is the sixth release of the language, and was released in June 2015. ECMAScript, or “ES” for short, is also generally referred to as “JavaScript”. There have been many new additions and changes made from JavaScript ES5 (the previous version) to ES6. Some of the changes that will be highlighted in this example will be constants, block-scope variables and functions, default parameter values, and string interpolation. Finally, there are several new built-in functions and formatting options.

1.1 Constants

One of the new features of ES6 is the ability to use constant variables. A constant variable is one that cannot be assigned any new content. Instead of using the typical `var` to declare the variable, `const` is used. `var` was the only option available in ES5, which meant that any variable created in the code could be changed at any other point in the code.

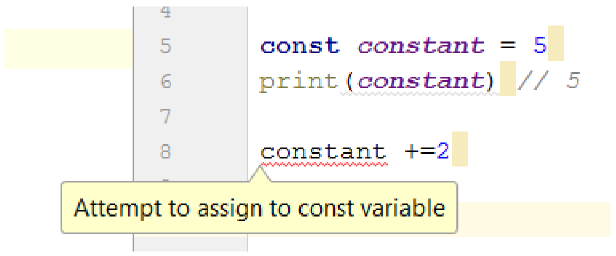
Listing 1: Const Declaration

```
const constant = 5
print(constant) // 5
```

Listing 2: Error, the const value can't be changed.

```
constant +=2
```

- With a const variable, this will not be allowed, and will pop up an error that indicates constant is of type const and cannot be reassigned.



```
const constant = 5
print(constant) // 5

constant +=2
```

Const can be especially useful in programming situations where there are multiple programmers or developers working on the same project. It makes the code a little easier to read, and lets other developers know that the variable will not be changing, and should not be changed by them. [PR_Newswire] [Simpson]

1.2 Block-Scope and Let

Block-scope variables are variables that can only be used inside a 'block' of code. With `var`, any variable declared in JavaScript with ES5 is a global variable, or one that can be accessed anywhere in the function. [Block_scope]

Listing 3: Global variable

```
var global = "Hello";

function block (x)
{
        var a = 5;
}

console.log(global);
console.log(a)
```

Because both variables were declared with `var`, they were global variables that could be called later in the program, as shown by the output below. This was the only available option in ES5, although `var`, and using global variables, is still used in ES6.

Listing 4: Output

```
Hello
5
```

ES6 has included an option to use `let` instead of `var` when declaring a variable, which will make the variable it will be a block-scope variable. The below code is similar to the above version, except that the `var a` is replaced by `let block`.

Listing 5: Block-scope variable

```
var global = "Hello";

```

(continues on next page)

(continued from previous page)

```
function block (x)
{
        let block = 5;
        console.log(block)
}

console.log(global);
console.log(block)
```

Listing 6: Output

```
5
Hello
Reference Error Exception
```

[Compatibility]

[ECMAScript_6]

[Prusty]

1.3 Parameter Values

Default parameters are used when the programmer wants a certain value to be set if one isn't given when the method is called. If a parameter is specified but not given a value, it is set to `undefined`.

Having an undefined answer when a function is called could cause errors, give an incorrect answer, or even crash the program. Programmers could find default parameters useful to help avoid these situations. ES5 did have a way to set default parameters, but it was slightly complex and time consuming. The new ES6 version is much easier to use, and makes the code nicer to read.

In ES5, there was no easy way to set default parameters. Instead, programmers would check within the function to see if the parameter was undefined and then set it to a value if it was.

- What was used in ES5

Listing 7: Return the sum of three numbers

```
function defaultValues(a, b, c)
{
        if (b ===undefined)
                b = 5;
        if (c === undefined)
                c = 12;
        return a + b + c;
}

f(1, 2, 3)

f(1, 2)

f(1)
```

- What is used in ES6 - simpler

Listing 8: Return the sum of three numbers

```
function defaultValues(a, b = 5, c = 12)
{
        return a + b + c;
}

f(1, 2, 3)

f(1, 2)

f(1)
```

- Output

Listing 9: The output of both functions remains the same.

```
f(1, 2, 3) === 6 //1+2+3
f(1, 2) === 15 // 1+2+12
f(1) === 18 //1+5+12
```

[Prusty]

[ECMAScript_6]

1.4 String Interpolation

ES6 adds an update the JavaScript's string interpolation. The first update that was made from ES5 to ES6 was the ability to write strings on multiple lines without having to program in concatenation at the end of each line. There actually was a way to "technically" accomplish this in ES5, but it was also considered a bug and not recommended to use.

Listing 10: Correct was to use String Interpolation in ES5

```
var string = "Here is a string \n" +
"on multiple line"
```

Listing 11: ES5 Bug

```
var string = "To get a string on multiple lines \"
"a programmer could put a backslash \"
"at the end of the line and the computer would read it \"
"all as one line"
```

ES6 String Interpolation also makes it easier for programmers to call attributes of objects in strings without having to use concatenation. Previously in ES5, in order to call an object attribute and add it to a string, the programmer would have to end the string and concatenate on the object's attribute. In ES6, this was changed so that the object call could be made within the string itself. This, in addition to being able to write strings on multiple lines made strings much easier to code, and easier for other programmers to read.

Listing 12: ES5 Concatenation

```
var person = {firstName = "Sara", lastName = "Brown", occupation = "student"}

```

(continues on next page)

(continued from previous page)

```
var college = {name = "Simpson College"}

var string = person.firstName + person.lastName + " is a " + person.occupation +", \n
↪" +
"at " + college.name + "."
```

Listing 13: ES6

```
var person = {firstName = "Sara", lastName = "Brown", occupation = "student"}

var college = {name = "Simpson College"}

var string = `${person.firstName} ${person.lastName} is a ${person.occupation}
"at ${college.name}.`
```

An important part of this change was that in order to signify a string that will be on multiple lines, or have an object selected in the middle of the string is by using ‘ *back ticks* ‘ instead of the normal “double quotes” or ‘single quotes’.

[Zakas_Understanding] pg 26-28 [es6_Features]

1.5 New Built-in Methods

Several built in functions for ES5 have been updated to work faster and/or be easier to read and code.

- **Repeating Strings** As the name suggests, this function allows the programmers to repeat a string a certain number of times.

  Listing 14: Es5

  ```
  Array(5).join("hello")
  ```

  Listing 15: Es6

  ```
  "Hello".repeat(5)
  ```

- **Searching in Strings** Searching in strings has also been updated in ES6 for simplicity and easier readability. It was possible to search strings in ES5, but the only method that was used was `.index`. `.index` was also a lot more complicated to use, and wasn’t as easily read through afterwards. The new methods in ES6 include `.startsWith`, `.endsWith`, and `.includes`.

  ```
  "Sara".startsWith("Sa")
  "Simpson".endsWith("son")
  "JavaScript".includes("Scr")
  //You can also specify where to start in the string
  "Simpson".startsWith("imp", 1)
  "Simpson".startsWith("imp", 2)
  ```

  Listing 16: Output

  ```
  true
  true
  true
  ```

(continues on next page)

(continued from previous page)

```
true
false
```

- **Number Type** In ES5, to check a number's type, the programmer would have to write a function themselves to do it. ES6 now includes several functions to help check number types. These methods include `.isNaN` which checks if something is not a number, and `.isFinite` which checks to make sure you have a finite, and not an infinite, number. Both functions are used by calling Number, then ".", then the name of the function that is wanted.

  For this testing, the variable Infinity is used. Numerical, JavaScript uses this to store a number that exceeds the upper limit of the floating point. If printed out, it would display "Infinity". If displayed as a number, it would show 1.797693134862315E+308. It can also be used to represent negative infinity by putting a "-" sign in front.

```
Number.isNan(2017)
Number.isNan(Hello)

//JavaScript has the variable Infinity which exceeds the upper limit of the
floating point.
Number.isFinite(Infinity)
Number.isFinite(-Infinity)
Number.isFinite(2018)
```

Listing 17: Output

```
true
false

false
false
true
```

- **Number Truncation** Number truncation is a pretty simple function, its purpose is to take a floating point number and drop off the decimal or fractional part. However, it does not round the number, it strictly drops off the decimal. Like Number Type, this was possible in ES5, but the code had to be written by the programmer and it was not a built in function.

Listing 18: ES6

```
console.log(Math.trunc(96.9)
console.log(Math.trunc(12.1)
console.log(Math.trunc(0.1)
```

Listing 19: Output

```
96
12
0
```

- **Number Sign** Number sign is also a simple function that takes place of the programmer having to personally write the function. This function will return what sign the number entered has. The possible answers are 1 (positive), -1 (negative) and 0/-0 for positive and negative 0 or decimal numbers

```
console.log(Math.sign(2017))
console.log(Math.sign(-2014))
```

(continues on next page)

(continued from previous page)

```
console.log(Math.sign(0))
console.log(Math.sign(-0.1)
```

Listing 20: Output

```
1
-1
0
-0
```

[ECMAScript_6]

1.6 New Formatting Methods

There have been several new updates that have been added to ES6 that are based on location. These include new formatting functions for time and date, currency, and money. They are all built in functions, and the location is based on a BCP 47 language tag. Some examples of a BCP 47 language tag included: [Arai]

- “hi” - Stands for Hindi
- “de” - Stands for German
- “en” - Stands for English

You can also add on locations in addition to language, in order to work with different dialects. For example:

- “en-US” is English in the United States
- “de-DE” is German in Germany
- “de-AT” is German used in Australia

All the new functions are first called using `Intl`, followed by the function name. This is used to set a variable to the specific language, or country dialect. To use this new formatting, the programmer will then go `variableName.format(Number to format)`.

1.6.1 The New Formatting Functions

- Number Formatting:

```
var american = new Intl.NumberFormat("en-US")
var german = new Intl.NumberFormat("de-DE")

german.format(999888777.58)
american.format(999888777.58)
```

german.format will return “999.888.777,58”, and the american.format will return “999,888,777.58”. The difference between the two may seem small, as the German number system uses commas were the American uses periods and vice versa, but it does create several benefits, such as

- Making it easier to format to local currency, as there was no easy way to do this in ES5
- Easier to reformat for use in different countries, as programmers and their developers and/or users can be global
- It would also be easier to read - countries may use similar signs but different decimal/commas, makes it easier to see which currency it’s referencing

- Currency Formatting:

The currency formatting starts off similar to the basic number formatter, but adds on a section that specifies “currency”, and what then what specific currency to use.

```
var american = new Intl.NumberFormat("en-US", {style: "currency", currency:
↪"USD")
var italian = new Intl.NumberFormat("it-IT", style: "currency", currency:
↪"EUR")

america.format(147258.36)
italian.format(147258.36)
```

Listing 21: Output:

```
$147,258.36

147.258,36€
```

- Date and Time Formatting:

Dates and times use a different function that NumberFormat, quite intuitively called `DateTimeFormat`. Similar to the first number formatter, all the needs to be put in the parentheses is the BCP 47 code. This is especially useful when translating dates that just switch the order of the day and month, as these could be easily confused. Three different examples of date formatting would be day/month/year (Germany), month/day/year (United States), and year/month/day (Japan).

```
var american = new Intl.DateTimeFormat("en-US")
var german = new Intl.DateTimeFormat("de-De")

american.format(new Date(2017-04-13))
german.format(new Date(2017-04-13))
```

Listing 22: Output:

```
4/13/2017

13.4.2017
```

There are no equivalent functions in ES5, so all of these functions are brand new to ES6. [ECMAScript_6]

1.7 Conclusion

There have been many different updates to the newest version of JavaScript, from fixing smaller functions to work better, adding in entirely new functions, or adding in different programming styles. Many of these updates give the programmer the option to write code that is either easier or more straight-forward than before, or simply make the code more readable.

1.8 Sources

CHAPTER 2

How-To Examples

Aside from the documentation for [sphinx], here are some examples that people ran into the most problems with when doing the writing last year. Citation Example ————-

For citations, we'll mostly try to follow the MLA format. You should include in-text citations. At the end of a phrase, paragraph, or section where you use the information, include the citation. [PurdueMLA]

This is a standard order for a text citation. [TextCitation]

If you use an autogenerator for your reference, watch for "nd" and "np". You Sometimes I see student citations that have both np and a publisher listed. That makes no sense. Sometimes the URL includes the date in it. You can use that.

Don't list "espn.com" as the publisher. Make it "ESPN". Search for an "about" page if you aren't sure who published.

Watch out for: Don't use Google as a source, unless it actually came from Google. Google indexes documents and images on the web. Find the original locaiton.

Watch out for: A URL is not a citation. Repeat after me. A URL is not a citation. Do not every, in this class or any other, use a simple URL as a citation.

2.1 Example

Wikipedia says that the Directory Traversal Attack [dta] is a kind of attack that involves traversing directories.

If I forgot how to do reStructuredText I could look at the Sphinx website [sphinx].

2.1.1 Code Samples

Need code in your program? Here's how.

2.2 In-line code sample

You can do an in-line code example of how a for loop works, such as `for (int i=0; i < 10; i++) {`, by surrounding it with back-tics.

2.3 In-document code sample

Here, I have a longer code block in the document.

Listing 1: Const Declaration

```
const constant = 5
print(constant) // 5
```

2.4 Including an external file

This loads a file OUTSIDE the document. I love this because I can run the file to make sure it works. I am also highlighting a line and adding line numbers.

Listing 2: example.js

```
var global = "Hello";

function block (x)
{
    var a = 5;
}

console.log(global);
console.log(a)
```

2.4.1 Image Examples

You can do images as a figure with a caption:

Or just as an image:

2.4.2 Call-outs

You can create your own call-outs.

Fig. 1: Corgi image from [freeclassifieds].

Warning: Make sure you match case with images! It may work on your computer, but it won't work when you deploy it.

But they don't have to be so angry.

Note: Only you can prevent forest fires. Really. Because we cut back on the budget and there isn't anyone else.

2.4.3 Roles

See: https://www.sphinx-doc.org/en/master/usage/restructuredtext/roles.html

You can hit `ctrl-c` to stop a running program.

To continue, hit *Start* → *Programs*

You can do math equations: $x = \frac{5}{a} + b^2$

#CHAPTER 3

Final Reports

3.1 React JS - STA

React is a declarative, efficient, and flexible JavaScript library. Created by facebook engineer Jordan Walke, it was meant to help build user interfaces with a focus on being fast and flexible. React has helped expand the way front end developers interact with various user interfaces, allowing them to make complex user interfaces in very small code sizes. This tutorial will introduce you to the basics of React and show you how to create a Tic Tac Toe game.

3.1.1 History of React

React started as a version of PHP, called XHP, that was created by Facebook. Engineer Jordan Walke wanted to be able to create dynamic applications on the web, so he found a way to implement XHP in a browser along with javascript. Very soon after, Facebook officially started using React, and it has grown in popularity in the following years [reactBackground].

React was first used by Facebook and has continued to grow over the years. In May 2013, Facebook announced that it would open source React and its popularity began to skyrocket. Fast forward to 2015 and many companies had begun to show real interest in React. Flipboard, Netflix, and AirBnB were just a few of the earliest supporters.

3.1.2 Fundamentals of React.js

Babel & JSX (Use references)

Babel is a JavaScript compiler that is mainly used to convert codes into compatible versions of JavaScript in different browsers. Some of the main uses of Babel include:

- Transforming Syntax
- Polyfill features that are missing in your target environment
- Source Code Transformations

JSX is a separate technology from React, and is completely optional in building a React Application. However, it does make everything much simpler when you combine the two.

React uses JSX because it is fast. JSX performs optimization while it compiles code. It is also type-safe, allowing errors to be caught during compilation rather than at runtime. Finally, it is easy to learn, especially for individuals who have worked with HTML [w3React].

Components

A component is a JavasScript class that may accept inputs (much like a Java function). This class then returns a React element telling how the user interface(UI) should look. In React, everything is considered a component. They are the building blocks of any app in React. Here is an example of a Greeting component:

```
const Greeting = () => <h1> Hello World! It is beautiful
today!</h1>
```

This component returns a greeting that prints out "Hello World! It is a beautiful day today!"

Handling Data (Props vs State)

In React, there are 2 different types of data. The first is a prop. Props give us the ability to write a component one time and then reuse it for different cases. Here is how we would pass props to a component:

```
    ReactDOM.render(
        <Hello message="Sam is cool" />,
        document.getElementId("root")
    );

This prop has a message with the value "Sam is cool". In order to access
this, we can reference 'this.props.message':
```

```
    class Hello extends React.Component {
        render() {
            return <h1>Hello {this.props.message}!</h1>;
        }
    }
This code would then produce a screen that prints out "Sam is cool"!
```

The second way of storing data in React is using the components state. This allows for the component to be able to change its own data. This is useful for when you want the data in your app to change based on something like user input.

```
class Hello extends React.Component {

    constructor(){
        super();
        this.state = {
            message: "Sam is (from state)!"
        };
        this.updateMessage = this.updateMessage.bind(this);
   }
    updateMessage() {
        this.setState({
            message: "Sam is (from changed state)!"
```

(continues on next page)

(continued from previous page)

```
        });
    }

    render() {
        <div>
        <h1>Hello {this.state.message}!</h1>
        <button onClick={this.updateMessage}>Click me!</button>
        </div>
    }
}
```

Here, we initialized state first, modified the state using updateMessage(), and added a button to call the updateMessage function. The button then changes the message value when we click it [learnReact].

3.1.3 Creating an Application in React

Lets look at a React Tutorial to create a Tic Tac Toe Game.

To begin, we are provided a starter code that styles our board using CSS and creates 3 components: Square, Board, Game.

The first thing we will have do is change the code in Board's renderSquare method, which will allow us to place a value on each square in the board. We will also change Square's render method to show the value in each square and fill it with an 'X' when we click it. (We will use the arrow function syntax () => for event handlers).

```
class Board extends React.Component {
    renderSquare(i){
        return <Square value={i} />;

class Square extends React.Component {
    render(){
        return(
            <button className="square" onClick={() => {
                alert('click'); }}>
                {this.props.value}
            </button>
        );
    }
}
```

Next, we will use state to help the Square component know that it got clicked and fill it with an "X". We will also change the Squares render method to show the state's value when we click it.

```
class Square extends React.Component {
      constructor(props) {
        super(props);
        this.state = {
          value: null,
        };
      }
      render() {
        return (
          <button
            className="square"
            onClick={() => this.setState({value: 'X'})}
```

(continues on next page)

(continued from previous page)

```
        >
          {this.state.value}
        </button>
      );
    }
  }
```

By calling `this.setState` from onClick, we tell React to re-render the Square when it's <button> is clicked.

Now, we want to be able to determine a winner. In order to do that, we need to add a constructor to the Board and make Board's starting state have an array of 9 nulls that correspond with the 9 squares of the board.

```
class Board extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      squares: Array(9).fill(null),
    };
  }
  renderSquare(i) {
    return <Square value={this.state.squares[i]} />;
    }
```

Each Square will not obtain a value of 'X', 'O', or null if it is empty.

Now, in order for the Square to update the Board when clicked by the user, we need to make a change in the renderSquare method of Board to include an onClick listener. We will also need to change the Square component to accept the two props from Board, `value` and `onClick`.

```
renderSquare(i) {
   return (
     <Square
       value={this.state.squares[i]}
       onClick={() => this.handleClick(i)}
     />
   );
 }
 class Square extends React.Component {
 render() {
   return (
     <button
       className="square"
       onClick={() => this.props.onClick()}
     >
       {this.props.value}
     </button>
   );
 }}
```

When you try and click a Square, you should get an error. This is because the `handleClick()` has not been defined yet in Board. Edit your code to look similar to this:

```
class Board extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      squares: Array(9).fill(null),
```

(continues on next page)

(continued from previous page)

```
    };
  }
  handleClick(i) {
    const squares = this.state.squares.slice();
    squares[i] = 'X';
    this.setState({squares: squares});
  }
  renderSquare(i) {
    return (
      <Square
        value={this.state.squares[i]}
        onClick={() => this.handleClick(i)}
      />
    );
  }
```

You should now be allowed to click the Squares to fill them with an input. This works because we are not storing the state in Squares, but sending it from Board which allows Square to re-render automatically. The Board has control over the Square components, which we can refer to as controlled components.

Ok by this point you're probably tired of reading all this code and making seemingly redundant changes! We're almost done!

We want to change Square to be a function component. These components are simpler for methods that only have a `render` method and dont have their own state. Change the Square class to look like this function:

```
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

Finally, we want to be able to take turns (alternate between X's and O's). By default we can set the first move to be "X".

```
class Board extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      squares: Array(9).fill(null),
      xIsNext: true,
    };
  }
```

The boolean at the end of the constructor, `xIsNext` needs to flip each time a user goes and stores the games state. We can edit this in Boards `handleClick()` function. In Board's `render` we will then change the "status" text to display what players turn it is.

```
handleClick(i) {
    const squares = this.state.squares.slice();
    squares[i] = this.state.xIsNext ? 'X' : 'O';
    this.setState({
      squares: squares,
      xIsNext: !this.state.xIsNext,
    });
```

(continues on next page)

(continued from previous page)

```
  }

  renderSquare(i) {
    return (
      <Square
        value={this.state.squares[i]}
        onClick={() => this.handleClick(i)}
      />
    );
  }
```

Lastly (I promise!!), we want to declare a winner after the game is over. Put this helper function at the end of the file to allow your program to calculate a winner.

```
function calculateWinner(squares) {
  const lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ];
  for (let i = 0; i < lines.length; i++) {
    const [a, b, c] = lines[i];
    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
      return squares[a];
    }
  }
  return null;
}
```

With the use of the `calculateWinner` function, we can replace the `status` in Board's `render` function. We can also now change Board's `handleClick` method to ignore a click if we have a winner, or that Square is filled already.

```
render() {
    const winner = calculateWinner(this.state.squares);
    let status;
    if (winner) {
      status = 'Winner: ' + winner;
    } else {
      status = 'Next player: ' + (this.state.xIsNext ? 'X' : 'O');
    }

handleClick(i) {
    const squares = this.state.squares.slice();
    if (calculateWinner(squares) || squares[i]) {
      return;
    }
    squares[i] = this.state.xIsNext ? 'X' : 'O';
    this.setState({
      squares: squares,
      xIsNext: !this.state.xIsNext,
```

(continues on next page)

(continued from previous page)

```
    });
  }
```

Now you should have a functional working tic tac toe game!! Hopefully you have now learned a little more about the basics of React and why it works. Here's a cleaned up version of the code I've shared: [TicTacReact] Now there's other functionality that could be added (storing history of moves, showing past moves etc), but that's for you to play with! However, this link will take you through some more of the code if you wish to explore further [reactTutorial].

3.1.4 What is the future of React?

React is a relatively new technology, only gaining popularity withing the last 5 years. With the amount of support React has and developers interested in using it, React will stick around for awhile. It's simplicity, and conciseness has shown that it definitely has its place in the programming world [futureReact]. Here are just a few of the companies that actively use React today:

- Facebook
- WhatsApp
- Uber
- Netflix
- Yahoo
- Sberbank(#1 bank in Russia)

3.1.5 Conclusion

As we have now learned, React is especially helpful for creating complex user interfaces. React makes it much simpler to write code for applications and has already become one of the most popular libraries for web development. With its popularity continually growing since its creation, it is hard to see React falling out of relevance. While it is not likely that it will reach the levels of other languages like Python or Java, React will be very resourceful for years to come.

3.1.6 Works Cited

3.2 ReactJS

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. Since React is fast and flexible, it allows developers to make complex UIs from small and isolated pieces of code called "components". Components are essentially JavaScript functions or classes that describe how certain segments of the user interface should look [reactIntro]. This article explains how React came to be, why people should learn it, and how to use it.

3.2.1 History

A software engineer at Facebook named Jordan Walke is the creator of React. Around 2010, Facebook struggled with code maintenance. They were implementing new features of Facebook Ads and it made the Facebook application increasingly complex. The complexity of the application caused Facebook to slow down as a company. They eventually ran into many cascading updates with their user interface, and software teams could not keep up. In 2011, Jordan Walke created the first prototype of React called FaxJS to make everything more efficient.

In 2012, React started being used by Facebook. Facebook also acquired Instagram. Instagram wanted to use Facebook's technology and this eventually led to React being open-sourced. Initially people thought React was a big step backward, but over time it grew in reputation. In 2014, Facebook started appealing to enterprises like Netflix as a selling point. Over the past few years React has grown immensely and has become a leading JavaScript library [reactHistory].

3.2.2 Popularity

React is arguably the most popular JavaScript library on the market right now. In June 2018, React was mentioned in over 28% of job postings across popular languages. Vue and Angular were far behind, with under 10% of job postings listing them. React also has significantly more Node.js Package Manager (NPM) downloads than Vue or Angular, which shows more people are using React for their websites than these other competitors [reactPopularity]. Popular websites using React are:

- Facebook
- Instagram
- Uber
- WhatsApp
- Khan Academy
- Netflix
- PayPal
- Airbnb
- and many more…

3.2.3 Advantages

Why are so many people using React compared to other JavaScript libraries? One reason is that it's very easy to use. Later, we will see how simple it is to implement React in a project. Another reason for its popularity is it breaks code down into reusable components. This makes code more maintainable and easier to change especially in larger projects. Along with technical advantages, since React has a large amount of users there are a lot of people ready to help when developers run into issues [reactPopularity].

3.2.4 Future

React is a relatively new technology that has exploded in the last five years. With React being by far the most popular JavaScript library used right now, I don't see it going away in the next five to ten years. Even if another better library comes along, it will take awhile for React to dwindle into obscurity. With React's community support and technical benefits for current technologies, it has a continuing bright future ahead.

3.2.5 About React

React has features that make it more powerful. It utilizes Babel and JSX, components, and unique data storage techniques. This section takes a look at these features.

What is Babel and JSX?

React uses something called Babel to translate JSX code into JavaScript. Babel is a JavaScript compiler that can translate markup or programming languages into JavaScript. JSX stands for JavaScript XML. It takes elements from XML, HTML, and JavaScript and combines it into one language [reactW3Schools]. Example JSX code looks something like this: `var element = <h1>This is a Header Variable!</h1>`

React Components

Almost all code using React is in components. Components are basically bite-sized pieces of code that perform one functionality. Components can be either JavaScript functions or classes. Inside components there is often a method called `render()`. The `render()` method is used to display HTML elements [reactSimple]. Components use two types of data storage called Props and State, which we will look at next.

Data Storage

Props and State are how React handles data. Props are essentially parameters passed into a component from a different component, while state is private and can only be changed within the component itself. If a component needs external data it will rely on props. Internal data will be controlled by state [reactSimple]. The difference between props and state will be shown more clearly in the later tutorial.

Best Practices

There are several helpful tips to know when using React that will make code cleaner and more efficient:

- It is good programming practice to put most of the application's logic in a component's `render()` method.
- It is best to avoid state as much as possible and pass data using props instead.
- When passing props into components the PropType should be defined to improve readability.
- Components should only be responsible for a single functionality.
- It is more maintainable to have many small components than a few large ones [reactBestPractices].

When Should React be used?

React is most helpful when building an advanced user interface. When developing simple, static web pages React is pointless. React makes complex interfaces easier to maintain and more efficient. By using JSX components, it is usually easier to write and change than JavaScript and other JavaScript libraries. React is also easy to learn and has a large community to help with developing issues [reactPopularity].

3.2.6 React Tutorial

This section will help explain components and data storage in React through simple examples. At the end, we will create a basic React application.

Setup

The following HTML code shows how to get React into a project. There are three head scripts, and than one script in the body that refers to the React JSX file.

Listing 1: Setup

```
<html>
    <head>
        <script src="https://unpkg.com/react@15/dist/react.min.js"></script>
        <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js"></script>
        <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></
↪script>
    </head>
    <body>
        <div id="root"></div>
        <script type="text/babel" src="reactCode.jsx"></script> <!-- refer to React␣
↪JSX file here -->
    </body>
</html>
```

Components

As mentioned before, React components can be either JavaScript functions or classes. In this section, we will make a simple component using both methods. It is important to note, however, that using classes for components is more common among React developers.

Listing 2: Simple Class Component

```
class Example extends React.Component {
    render() {
        return <h1>I am a simple React component!</h1>;
    }
}

ReactDOM.render(
    <Example />,
    document.getElementById("root")
);
```

Listing 3: Simple Function Component

```
function Example(){
  return <h1>I am a simple React component!</h1>;
}

ReactDOM.render(
    <Example />,
    document.getElementById("root")
);
```

Awesome! We now have a working React component! Now let's take a look at using props and state in React components.

Data Storage

Data can be used in React using props or state. The following code shows how to use props:

Listing 4: Props

```
class Example extends React.Component {
        render() {
            return <h1>Hi, my name is {this.props.name}!</h1>;
        }
    }

ReactDOM.render(
    <Example name="Edward"/>,
    document.getElementById("root")
);
```

Notice how the data for the Example component is passed in from outside the component itself. Props cannot be changed once inside the component. To change data inside a component, state needs to be used. Here is a simple example of using state:

Listing 5: State

```
class Example extends React.Component {

    constructor(){
        super();
        this.state = {
            name: "Lukas"
        };
    }

    render() {
        return <div><h1>Hi, my name is {this.props.name}!</h1><br></br>
               <h1>Hi, my name is {this.state.name} and I'm from state!</h1></div>;
    }
}

ReactDOM.render(
    <Example name="Edward"/>,
    document.getElementById("root")
);
```

Great! Now that we have learned components and data storage, let's make a simple application that takes a name input and prints it out on the screen.

3.2.7 Simple Application

For this application, we are going to make a few changes to our Example component. We first need to change our `render()` method to display a name input and button.

Listing 6: Render Method

```
render() {
        return (
            <div>
                <label>
                  Name:
                  <input type="text" value={this.state.name} onChange={this.
↪changeName} />
```

(continues on next page)

(continued from previous page)

```
                </label>

                <button type="button" onClick={this.submitName}>Submit</button>
                <br></br>

                <h1>My name is {this.state.submittedName}!</h1>
            </div>
    );
}
```

Next, we need to change the constructor of our component to use prop data and bind "this" to the functions we will create. Without binding the "this" keyword to the functions, we would not be able to access "this" within the functions. The two simple functions simply set state data.

Listing 7: Constructor and Functions

```
constructor(props){
    super(props);
    this.state = {
        name: props.name,
        submittedName: props.name
    };

    this.submitName = this.submitName.bind(this);
    this.changeName = this.changeName.bind(this);
}

submitName(){
    this.setState({submittedName: this.state.name});
}

changeName(event){
    this.setState({name: event.target.value});
}
```

Nice work, we are finished! Here is what the end result should look like:

Listing 8: Final HTML Page

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <script src="https://unpkg.com/react@15/dist/react.min.js"></script>
        <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js"></script>
        <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
    </head>
    <body>
        <div id="root"></div>
        <script type="text/babel" src="reactCode.jsx"></script> <!-- refer to React␣
↪JSX file here -->
    </body>
</html>
```

Listing 9: Final JSX File

```
class Example extends React.Component {

    constructor(props){
        super(props);
        this.state = {
            name: props.name,
            submittedName: props.name
        };

        this.submitName = this.submitName.bind(this);
        this.changeName = this.changeName.bind(this);
    }

    submitName(){
        this.setState({submittedName: this.state.name});
    }

    changeName(event){
        this.setState({name: event.target.value});
    }

    render() {
        return (
            <div>
                <label>
                  Name:
                  <input type="text" value={this.state.name} onChange={this.
↪changeName} />
                </label>

                <button type="button" onClick={this.submitName}>Submit</button>
                <br></br>

                <h1>My name is {this.state.submittedName}!</h1>
            </div>
    );
    }
}

ReactDOM.render(
    <Example name="Edward"/>,
    document.getElementById("root")
);
```

3.2.8 Conclusion

React is a helpful JavaScript library when creating complex or dynamic user interfaces. Since code is in small chunks, React makes applications more maintainable and easier to write. Even though React is not a decade old, it is already the most popular JavaScript library for web development. With its technical benefits and large community support, I do not see React going away anytime soon.

`final/becker/result.PNG`

Fig. 1: Final Application Result

3.2.9 Sources

3.3 Vue

3.3.1 This is Vue

Vue.js is a progressive open-source JavaScript framework built for the purpose of building user interfaces. The Vue.js library is designed to be easily integrated with other libraries and existing projects. Vue.js architecture focuses on declarative rendering and component composition which we will get into in the later sections. [VueWiki] [VueIntroduction]

To include Vue.js within an HTML document, add the following script: `<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>`

Note that this is the development version of Vue.js. [VueIntroduction]

3.3.2 History of Vue

Vue was created and released in February of 2014 by Evan You [VueWiki]. You had formally worked for Google in Google's Creative Lab. He heavily used Angular 1 working on different projects and found that many of the features he either did not like or hardly used [Egghead]. Out of this, Vue was born.

You built a templating library for his own personal use and later released it as Vue.js. As the user community grew and additional features were added, Vue.js transformed from a small templating library into the open-source JavaScript framework that it is today. It is comparable to Angular which it grew out of [Egghead].

3.3.3 Declarative Rendering

The Vue.js system allows users to declaratively render data to the Document Object Model (DOM). From the surface, it appears like it is rendering a string template. However, Vue has done a lot of the work behind the scenes. The data and the DOM have been linked making everything reactive [VueIntroduction]. Let's take a look at an example to get a better understanding.

HTML Example

Listing 10: Vue Example HTML

```
<!DOCTYPE html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Vue Page</title>
</head>

<body>

<div id="app">
    <h1>{{ title }}</h1>
    <h2>{{ author }}</h2>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script type="text/javascript" src="vue.js"></script>
</body>
</html>
```

JavaScript Example

Listing 11: Vue Example JavaScript

```
    var app = new Vue({
        el: '#app',
        data: {
            title: 'Moby Dick',
            author: 'Herman Melville'
        }
    })
```

What does it mean for everything to be reactive? Run the above HTML and open the browser's JavaScript console. You can now set title and author to different values by typing `app.title = "Something"` or `app.author = "SomethingElse"`. The text should then render to whatever you set the new value to be.

Vue Directives

You have probably noticed that double braces `{{ }}` are used as place-holders for the data that is rendered from the JavaScript. With Vue.js, you can also call directives, which are HTML attributes, with the prefix `v-` [w3schoolsVue]. The `v-` prefix indicates that the directive is a special attribute provided by the Vue.js library. The examples below will walk you through a few examples of different Vue.js directives.

HTML Example

Listing 12: Vue Directive Example HTML

```
<div id="app">
    <p>{{ message }}</p>
    <p><input v-model="message"></p>

    <span v-bind:title="secretMessage">
        Hover over me!
    </span>
</div>
```

JavaScript Example

Listing 13: Vue Directive Example JavaScript

```
    var app = new Vue({
        el: '#app',
        data: {
            message: 'Hello there',
            secretMessage: 'This is a secret message'
        }
    })
```

This example shows the v-model and the v-bind directive. Like before, everything is reactive and both the message and the secretMessage can be changed with `app.message = "Something"` or `app.secretMessage = "Something"`.

The `v-model` directive creates a textbox for the user to interact with. In the above example, the `v-model` directive is tied to the `{{ message }}` place-holder. Whatever the user types into the textbox changes what the user sees above the textbox. The `v-model` directive is great for working with user input.

The `v-bind` directive binds an HTML element to a Vue instance. In the above example, `title` is bound to the vue instance of `secretMessage`. Whenever the user hovers over the title, the value of secretMessage appears to them.

These are just a few examples of the many Vue.js directives. We will work with a few more directives in the examples below.

Conditionals and Loops

Using directives, Vue gives you the ability to write "if" statements and "for" loops with `v-if` and `v-for`. The following example walks you through how to do conditionals and loops in Vue.js.

HTML Example

Listing 14: Vue Loops and Conditionals Example HTML

```
<div id="app">
    <p v-if="happy">Hello there friend!</p>
    <p v-else>Go away.</p>
    <button v-on:click="changeMood">Change Mood</button>

    <p>Grocery List</p>
    <ol>
        <li v-for="groceries in foods">
            {{groceries.text}}
        </li>
```

(continues on next page)

(continued from previous page)

```
    </ol>
</div>
```

JavaScript Example

Listing 15: Vue Loops and Conditionals Example JavaScript

```
    var app = new Vue({
        el: '#app',
        data: {
            happy: true,
            foods: [
                {text: 'Bread'},
                {text: 'Milk'},
                {text: 'Spinach'}
            ]
        },
        methods:{
            changeMood: function(){
                this.happy = !this.happy;
            }
        }
    })
```

The `v-if` directive and `v-else` directive, as you could guess, got together to allow you to do if statements and if-else statements. In the above example, the directive checks the value of the boolean variable `happy` and sets the text accordingly.

The `v-for` directive, as you would assume, allows you to do a for loop. In the example above, the for loops runs through the elements in `food` and displays them on to the page.

3.3.4 Component Composition

Another important concept of Vue is the Component System. The Component System is this abstract idea that you can build large scale applications with small, self-contained, and reusable parts. [VueIntroduction] Let's take a look at an example.

HTML Example

Listing 16: Vue Components Example HTML

```
<div id="componentsExample">
    <button-counter></button-counter>
    <button-counter></button-counter>
    <button-counter></button-counter>
</div>
```

JavaScript Example

Listing 17: Vue Components Example JavaScript

```
    Vue.component('button-counter', {
        data: function () {
            return {
                count: 0
```

(continues on next page)

(continued from previous page)

```
            }
        },
        template: '<button v-on:click="count++">You clicked me {{ count }} times.</
↪button>'
    })

    new Vue({ el: '#componentsExample' })
```

In the above example, we have created a component called `button-counter` in the JavaScript code. This component creates a button that keeps track of how many times it has been pushed. In the HTML code, the component is called upon three times which creates three separate buttons with the same function. Each button keeps track of its own count and not the overall count.

3.3.5 Conclusion

This has been a short introduction to Vue.js which has shown you some of the key attribute of Vue.js. Declarative rendering makes the Document Object Model(DOM) reactive to the data. Each time the data is changes, the DOM is updated as well. Component composition is another big attribute of Vue.js. Components allow you to make large scale applications with small, reusable parts.

3.3.6 Citation

3.4 Node.js

3.4.1 Introduction

Most websites and web applications must implement some form of a client-server network for them to have any meaningful functionality. There are many languages and technologies that allow you to implement this. However, if you wanted to create this application using JavaScript then you run into an issue. JavaScript only handles client-side programming. If you wanted to create a client-server network using JavaScript, you would have to write the server-side code in Java, or some other language. Node.js solves this issue by bringing JavaScript to the server-side. Node.js has an interesting history, and comes with a plethora of features and design choices that make it a scalable and efficient runtime environment for web development.

3.4.2 History

The most similar predecessor to Node.js was called, “NetScape LiveWire”. Unfortunately, there wasn’t a large demand for server-side JavaScript at the time. As a result, NetScape LiveWire was ultimately unsuccessful. As JavaScript became more advanced and efficient, the demand for server-side JavaScript capabilities also increased. This lead to the introduction of Node.js in 2009, created by Ryan Dahl [NodejsDev].

Node.js allows developers to create server-side JavaScript through the Node.js runtime environment. Shortly after its creation, other important libraries and features were created. Npm was created in 2009, and both Express and Socket.io were created in 2010. Node.js would continue to be updated with a new stable version every year, with Node.js 8 being released in 2017 and Node.js 10 being released in 2018 [NodejsDev]. The odd numbered Node.js versions are considered to be betas and the even numbered versions are considered as stable builds [LearningNode].

3.4.3 An Introduction to Node.js

What is it?

Node.js is a runtime environment that brings JavaScript to the server-side. It allows you to create web applications using nothing but JavaScript. Many developers are experienced with JavaScript and client-side programming. However, they may not be experienced with languages that support server-side programming like Java. This allows these developers to move to server-side programming without changing languages [Nodejs].

Node.js has other advantages as well. For example, Node.js is single-threaded. Multi-threaded networking tends to be less efficient and is difficult to implement. Since Node.js is single-threaded, it's far less likely to have thread-related bugs or issues. Despite being single-threaded, Node.js will never lock because everything is asynchronous. This also allows Node.js to handle thousands of requests at the same time [Nodejs].

Node.js runs on the Chrome V8 JavaScript engine. This makes it run very quickly, even for large-scale applications. Node.js has a massive collection of libraries that can be easily installed through npm. Some popular examples of these libraries are express, socket.io, koa, Colors, and more. It allows you to easily create and implement your own modules as well [Nodejs].

How does it work?

Unlike traditional programming, Node.js doesn't run line by line. Instead, Node.js relies on something called, "Asynchronous Programming". This isn't a new concept introduced by Node.js. However, Node.js uses it nearly exclusively [LearningNode]. The following jQuery code is an example of Asynchronous Programming.

Listing 18: jQuery on click

```
$('#example').on("click")
```

This program doesn't stall while waiting for that function to call. Rather, it calls that function when the event actually happens. That is how Node.js works. It is entirely event-based and relies on functions like the example above. Asynchronous Programming is advantageous because as mentioned earlier, it will never stall. It allows the website the process multiple things concurrently and supports live updates. To understand how Node.js implements Asynchronous Programming, consider the following, "Hello World!" example [LearningNode].

Listing 19: Hello World

```
var http = require('http');

http.createServer(function(req, res) {
        res.writeHead(200, {'content-type': 'text/plain'});
        res.end("Hello world!\n");
}).listen(8124);
```

So what does each line of this program actually do?

- **`var http = require('http');`** This loads the HTTP module which is essential for basic HTTP functionality and network access.
- **`http.createServer(function(req, res)`** This is a function within the HTTP module that creates a basic server. An anonymous function is passed in the parameter with the arguments of *req* and *res* which represent a server request and a server response. This function doesn't need to be anonymous.
- **`res.writeHead(200, {'content-type': 'text/plain'});`** This modifies the content type and status code of the response.

- **`res.end("Hello world!\n");`** This line writes, "Hello world!" and ends the response. Alternatively, you could do the following for the same effect.

  `res.write("Hello world!\n");`

  `res.end();`

- **`.listen(8124);`** This last line is an example of asynchronous programming. It's asynchronous because it only calls when the connection to the port is established.

[LearningNode]

The Event Loop

So how do these asynchronous functions actually work? Node.js relies on callbacks and operates with something called, "The Event Loop". The Event Loop operates in a series of phases. These phases can best be described with the following graph [EventLoop].

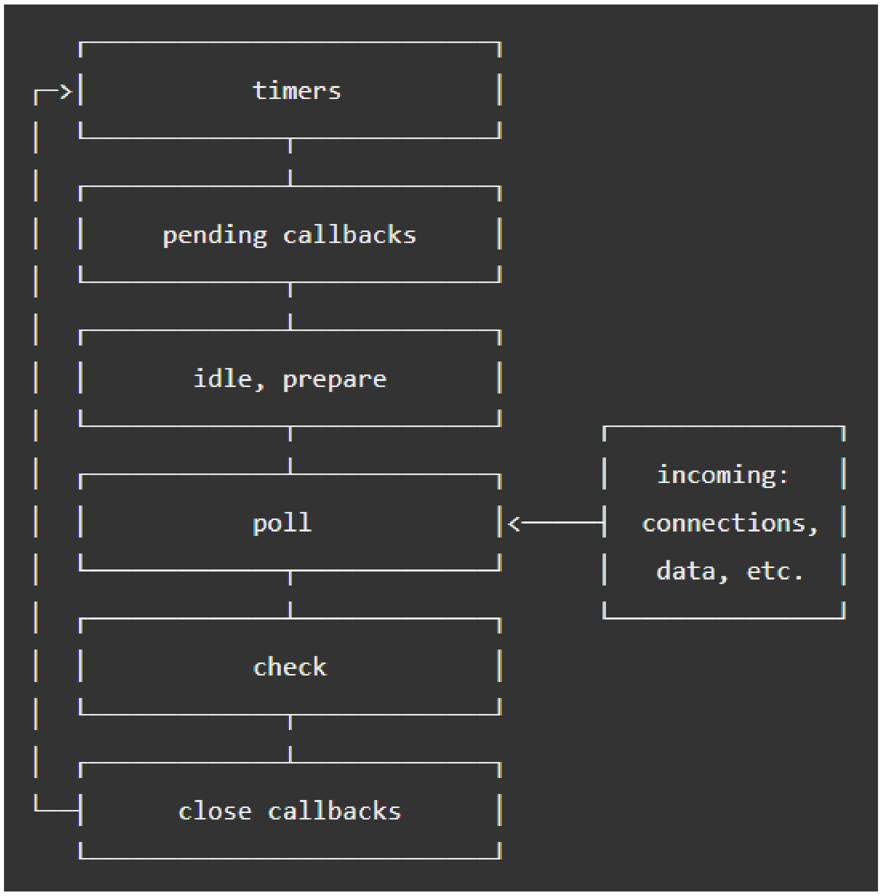


Fig. 2: [EventLoop]

So, the data starts by going to the *poll* phase which simply receives input or data. Next, it goes to the *check* phase which is where `setImmediate()` callbacks are executed. Then any callbacks involving disconnections or closings are called in the *close callbacks* phase. Then it runs all the callbacks defined by the timers in the *timers* phase. For example, through `setTimeout()` or `setInterval()`. Lastly, it runs any other callbacks that are still pending in the *pending callbacks* phase. There's also the *idle, prepare* phase but these can't be influenced since they run internally [EventLoop].

Timers can be used to delay the execution of some function. The most common way to accomplish this is through `setTimeout()` or `setInterval()`. The difference between these two is that `setTimeout()` runs once after the designated time, and `setInterval()` will continue running indefinitely with the interval as the designated time. For example, the following code will print out, "Hello World!" to the Node.js server console 1000 milliseconds after the program starts [LearningNode].

Listing 20: Hello World

```
function hello(res) {
        console.log("Hello World!");
}

setTimeout(hello, 1000);
```

The following uses `setInterval()` instead, in this example it prints out, "Hello World!" every 1000 milliseconds indefinitely.

Listing 21: Hello World

```
function hello(res) {
        console.log("Hello World!");
}

setInterval(hello, 1000);
```

[LearningNode]

What are its disadvantages?

Node.js has many advantages. However, it also has some issues. One major issue is that Node.js is not designed for computationally expensive applications. For example, it would not work well for optimization problems, or a GPS navigation application that calculates the best path to a destination. It's better to use Node.js for lightweight applications that have a lot of clients at once, such as chat rooms [LearningNode].

Since Node.js is asynchronous, it comes with an additional problem sometimes referred to as, "The Pyramid of Doom". This happens when there's an excess of nested callbacks that leads to an unreadable mess. However, "The Pyramid of Doom" can be easily fixed by having callbacks call outside functions rather than putting the code inside the actual callback. An even better solution is to use the Waterfall feature of the Async module. This feature works by chaining these callbacks together in an array [LearningNode].

3.4.4 Modules of Node.js

There are a plethora of modules you can install with npm. Each of them have their own unique features and uses. Developing Node.js applications is much easier through the use of modules. For example, some of the most important ones are socket.io and express. These modules greatly simplify the process of writing Node.js applications.

Socket.io

Socket.io makes communication between the server and its clients easier. Socket.io can also be used with Express which will be mentioned in a later section. How Socket.io works can be seen with the following code. This comes from the example project found at the end [SocketIo].

Listing 22: socket.io implementation

```
// This runs when a user connects to the server.
io.on('connection', function(socket) {
        // Some omitted stuff...

        // Tell the client (and only this client, that's why it's io.to) that they␣
↪connected to the server.
        io.to(socket.id).emit('on page loaded', picture);

        // More omitted stuff...

        // Runs when a user leaves the chat.
        socket.on('disconnect', function() {
                // Send a message to all clients that someone left.
                io.emit('chat message', "", "", 'red', users[socket.id].username + "␣
↪has left the chat.");
                io.emit('userlist remove', users[socket.id]); // remove them from the␣
↪user list
                users[socket.id] = null; // remove their information from the server
        });
});
```

As you can see, socket.io supports a large variety of functions such as connection, disconnect, emit, to, and more. These functions make it easier to transfer data between the server and clients [SocketIo].

Express

Express is a framework that makes developing Node.js faster and easier. In many ways, it's similar to how jQuery makes JavaScript development easier. Express is arguably the most widely used module that exists for Node.js. Because of this, many other modules such as Socket.io have support for Express. The following code is from the example project and demonstrates how Express can make Routing far easier. (Routing refers to the HTTP verbs, GET, PUT, DELETE, and POST. [LearningNode])

Listing 23: Routing in Express

```
app.get('/', function(req, res) {
        res.sendFile(__dirname + '/index.html');
});
```

This code determines how the server should respond upon receiving a GET request from index.html [LearningNode].

3.4.5 Node.js Development

REPL (Read-Eval-Print Loop)

There's a more efficient way to develop or test Node.js by using an interactive component called, "REPL". You can start REPL by simply typing, "node" into the Node.js console. REPL is very similar to the console found in Google Chrome's development console. You can simply type in some JavaScript and run it, with no need to mess with files. The following example demonstrates how REPL works. [LearningNode]

Listing 24: Example Node.js console using REPL

```
> var example = 10;
undefined
> console.log(example);
10
undefined
>
```

The symbol > designates a new line. So, in this example the user enters `var example = 10;`, and later types `console.log(example);` in another line which returns the result of the variable `example`.

REPL is a great tool for debugging and helps you figure out what's happening with your Node.js code. It also supports various libraries such as rlwrap which allows you to change the color of the text, along with other useful modifications. REPL also has custom commands such as .save which saves your REPL session into a file, allowing you to develop entire projects in REPL. Some other commands include the following. [LearningNode]

Command	Description
.break	Resets multi-line entries.
.clear	Resets everything.
.exit	Exits REPL.
.help	Lists all REPL commands
.load	Loads a previously saved REPL session.

Node.js also supports the capability to create custom REPL. This can be done in a normal Node.js file with `var repl = require("repl");`. Then you can create the custom REPL by using `repl.start`. REPL will execute everything defined in `repl.start` after running the file. [LearningNode]

Try to save often while using REPL.

Example Project

This is a project I created with Node.js, Socket.io, and Express. It's called, "Public Pixel Art" and is a web application that allows multiple people to all connect to a single chat where they can draw pixel art on the same canvas.

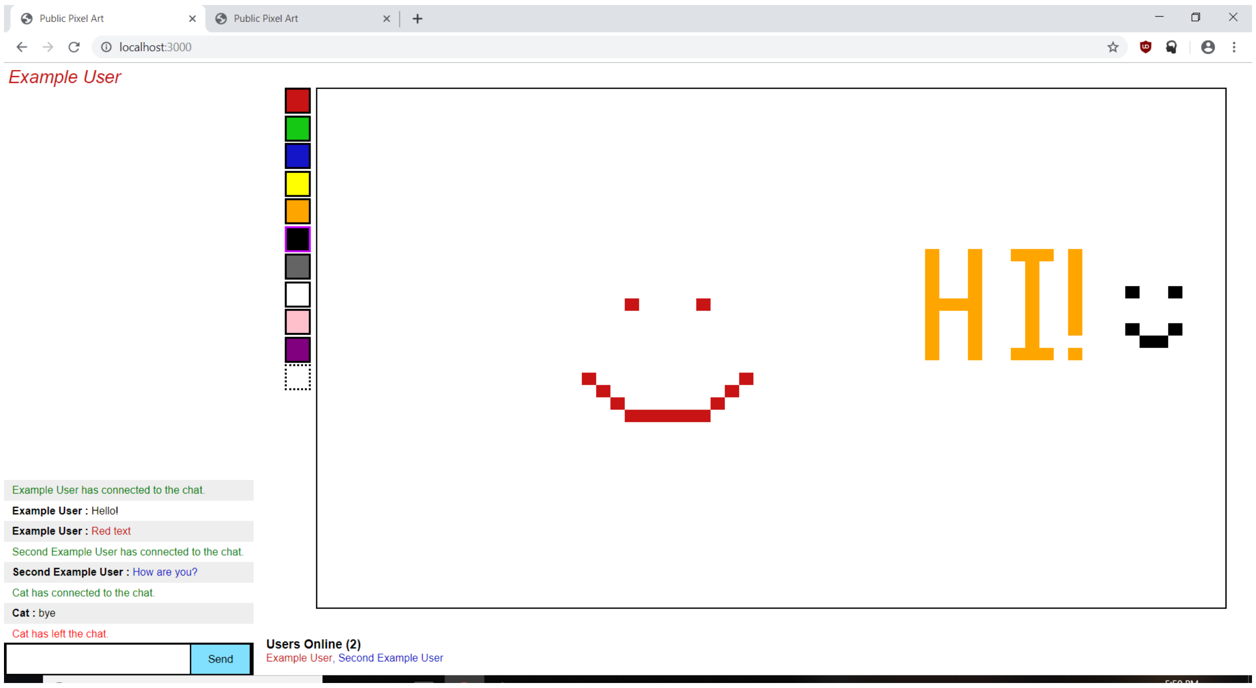
The CSS for the chat in the bottom left is a heavily modified version of the example from [SocketIo]

The following is the client-side code for the HTML page.

Listing 25: Client Side Code for Public Pixel Art

```
<!-- Public Pixel Art -->
<!--
    A web app that allows random people to connect to a chat where they can all draw␣
↪pixel art on the same canvas.
-->

<!doctype html>
<html>
    <head>
        <title>Public Pixel Art</title>
        <style>
            /*
                The CSS for the chat is a modified version of the example from https:/
↪/socket.io/get-started/chat/
```

(continues on next page)



Example User

Example User has connected to the chat.
Example User : Hello!
Example User : Red text
Second Example User has connected to the chat.
Second Example User : How are you?
Cat has connected to the chat.
Cat : bye
Cat has left the chat.

Users Online (2)
Example User, Second Example User

(continued from previous page)

```
            */
            * { margin: 0; padding: 0; box-sizing: border-box; }
            body { font: 13px Helvetica, Arial;}
            .chat_input { background: #000; padding: 3px; position: fixed; bottom: 0; width: 20%;}
            .chat_input input { border: 0; padding: 10px; width: 75%; margin-right: .5%;  }
            .chat_input button { width: 24%; background: rgb(130, 224, 255); border: none; padding: 10px;  }
            #messages { list-style-type: none; margin: 0; padding: 0;  }
            #messages li { padding: 5px 10px;  }
            #messages li:nth-child(odd) { background: #eee;  }
            #messages { margin-bottom: 40px; margin-right: 10%;  }
            .username_input { text-align: center; margin-top: 100px;}
            #users_online_section {margin-left: 21%; position: fixed; bottom: 2%;}
            #pic_frame {margin-left: 25%; margin-top: 2%; position: fixed; width: 73%;  height: 85%; border: 2px; border-style: solid;}
            #message_section {position: fixed; right: 77.8%; left: 0; bottom: 0;}
            #color_selection {position: fixed; width: 32px; margin-left: 22.5%; margin-top: 2%; list-style-type: none;}
            .color_option {width: 32px; height: 32px; border: 2px; border-color: black; border-style:solid; margin-bottom: 2px;}
            #username_disp {position: fixed; margin: 5px; font: 22px Helvetica, Arial;  font-style: italic;}
        </style>
    </head>
    <body id="body_stuff">
        <script src="https://cdn.socket.io/socket.io-1.2.0.js"></script>
        <script src="https://code.jquery.com/jquery-1.11.1.js"></script>

        <!-- Username in top left. -->
        <p id="username_disp">
```

(continues on next page)

(continued from previous page)

```
        </p>

        <!-- The messages from the chat. -->
        <div id="message_section">
            <ul id="messages"></ul>
        </div>

        <!-- List of users online. -->
        <div id="users_online_section">
            <h3 id="users_online_title">Users Online</h3>
            <p id="users_online" style="color: blue;"></p>
        </div>

        <!-- Input box for chat. -->
        <form id="chat_form" class="username_input" action="">
            <input id="chat_box" autocomplete="off" /><button id="sendButton">Set␣
↪Username</button>
        </form>

        <!-- Allows you to change the color you draw with. -->
        <ul id="color_selection">
            <li id="first_color" class="color_option" style="background-color:␣
↪rgb(200, 20, 20); border-color: rgb(200, 0, 255);"></li>
            <li class="color_option" style="background-color: rgb(20, 200, 20);"></li>
            <li class="color_option" style="background-color: rgb(20, 20, 200);"></li>
            <li class="color_option" style="background-color: yellow;"></li>
            <li class="color_option" style="background-color: orange;"></li>
            <li class="color_option" style="background-color: black;"></li>
            <li class="color_option" style="background-color: rgb(100, 100, 100);"></
↪li>
            <li class="color_option" style="background-color: white;"></li>
            <li class="color_option" style="background-color: pink;"></li>
            <li class="color_option" style="background-color: purple;"></li>
            <li id="custom_color" class="color_option" style="background-color: white;
↪ border-style: dotted;"></li>
        </ul>

        <!-- Pixels for the picture. -->
        <table id="pic_frame" border="0" cellpadding="0" cellspacing="0">
        </table>

        <script>
            $(function() {
                var socket = io(); // necessary to use socket.io
                var chosen_color = "rgb(200, 20, 20)"; // color you draw with
                var last_color = $('#first_color'); // used for highlighting the␣
↪currently selected color
                var choosing_name = true; // if the user has not yet set a username␣
↪this is set to true
                var user_color = "rgb(0, 0, 0)"; // color of the user's name and chat␣
↪color
                var users = 0; // total number of users

                // If the user clicks their username in the top left, change their␣
↪user color.
                // The user color changes the color of their text in chat and the␣
↪color of their username.
```

(continues on next page)

(continued from previous page)

```
                $('#username_disp').click(function() {
                    user_color = chosen_color; // Set user color to currently selected color
                    $(this).css("color", user_color); // Change color of username to new user color.
                    socket.emit("update user color", user_color); // Update new user color to server.
                });

                // Change which color the user draws with.
                $('#color_selection').on('click', 'li', function() {
                    // Highlight the currently selected color.
                    last_color.css("border-color", "rgb(0, 0, 0)");
                    last_color = $(this);
                    $(this).css("border-color", "rgb(200, 0, 255)");

                    /* If the user has selected the custom color option, have them input the RGB values for the custom
                       color. Otherwise, simply set their chosen color to the color they selected.*/
                    if ($(this).attr("id") === 'custom_color') {
                        var temp_color = prompt("Type in the RGB value for the custom color.");

                        // some redex to make sure they are inputting RGB values, supports 3 formats:
                        // rgb(r, g, b)
                        // (r, g, b)
                        // r, g, b
                        var format_a = /^[r][g][b][(]\d{1,3}[,]\s?\d{1,3}[,]\s?\d{1,3}[)]$/;
                        var format_b = /^[(]\d{1,3}[,]\s?\d{1,3}[,]\s?\d{1,3}[)]$/;
                        var format_c = /^\d{1,3}[,]\s?\d{1,3}[,]\s?\d{1,3}$/;

                        var valid = false;

                        // If the text matched format b or format c it can easily be converted to format a.
                        // The colors in the picture array are stored in format a so these must be in format a.
                        if (format_a.test(temp_color)) {
                            chosen_color = temp_color;
                        } else if (format_b.test(temp_color)) {
                            chosen_color = "rgb" + temp_color;
                        } else if (format_c.test(temp_color)) {
                            chosen_color = "rgb(" + temp_color + ")";
                        } else {
                            chosen_color = $(this).css("background-color");
                            alert("Invalid RGB format. Please enter the color with a valid format. Example: (200,50,50)");
                        }

                        // Update color of custom color box to the new color.
                        $(this).css("background-color", chosen_color);
                    } else {
                        chosen_color = $(this).css("background-color");
                    }
```

(continues on next page)

(continued from previous page)

```
                });

                // Tell the server which pixel the user clicks.
                $('#pic_frame').on('click', 'td', function() {
                    // pic_frame is a table of pixels, jquery can find td that was␣
↪clicked
                    var box_clicked = $(this).attr("id");
                    // The ids of the td elements are in a format axb (example,␣
↪20x30) where a and b is the column and row
                    var coords = box_clicked.split("x"); // find these coordinates by␣
↪doing a split with delimiter x

                    // Put the location the user clicked and their chosen color into␣
↪an object to send to server so that
                    // the picture may be updated on the server.
                    var click_info = {}
                    click_info.x = coords[0] || 0;
                    click_info.y = coords[1] || 0;
                    click_info.color = chosen_color;

                    socket.emit('box clicked', click_info);
                });

                // Runs when you click the submit button for the chat.
                $('#sendButton').click(function() {
                    socket.emit('chat message', $('#chat_box').val()); // send␣
↪information user typed to server
                    $('#chat_box').val(""); // empty chat box for new messages
                    return false; // prevents that weird refresh thing from happening
                });

                // Runs when a user loads the page.
                socket.on('on page loaded', function(picture) {
                    if (choosing_name) { // only run this for users who have not␣
↪selected their username yet
                        // Hide everything until they input their username.
                        $('#users_online_section').hide();
                        $('#pic_frame').hide();
                        $('#color_selection').hide();

                        // Set up the html for the picture from the server.
                        $('#pic_frame').empty();

                        for (var w = 0; w < picture.length; w++) {
                            var temp_row = $('<tr>');

                            for (var h = 0; h < picture[0].length; h++) {
                                var temp_cell = $('<td class="picbox" style=
↪"position: relative;">');
                                temp_cell.attr("id", w + "x" + h);
                                temp_cell.css("background-color", picture[w][h]);

                                temp_row.append(temp_cell);
                            }

                            $('#pic_frame').append(temp_row);
                        }
```

(continues on next page)

(continued from previous page)

```
                    }
                });

                // Update the color of the pixel the user clicked.
                socket.on('update box', function(box_info) {
                    var selector = box_info.x + "x" + box_info.y;
                    $('#' + selector).css("background-color", box_info.color);
                });

                // Display the message the user sent in chat.
                socket.on('chat message', function(username, msg, color, override) {
                    /*
                        Appending the html directly like this,
                        $('#messages').append(username + " : " + msg) etc
                        would allow clients to run javascript and html on other␣
↪clients, which is an issue.
                        All you would have to do is type something like this in chat,
                        <script>alert('hello');<//script>
                        and it would run that on every client. So, I'm appending it a␣
↪different way to prevent
                        this from happening.
                    */

                    // The override feature is used for messages sent by the server␣
↪instead of a user.
                    // For example, when someone leaves or joins.

                    if (override.length > 0) {
                        var temp_li = $('<li>');
                        temp_li.css("color", color);
                        temp_li.text(override);

                        $('#messages').append(temp_li);
                    } else {
                        // Display the user's message, automatically put in their␣
↪username and set the color of the
                        // message to their user color.
                        var temp_li = $('<li style="color: black;">');
                        temp_li.append($('<b>').text(username + " : "));

                        var temp_span = $('<span>');
                        temp_span.css("color", color);
                        temp_span.text(msg);
                        temp_li.append(temp_span);

                        $('#messages').append(temp_li);
                    }

                    // If there are too many messages, start removing them from the␣
↪top so they don't go off the screen.
                    if ($('#messages li').size() > 27) $('#messages li').first().
↪remove();
                });

                // After the user inputs their username, show everything and give␣
↪them permission to use everything.
                socket.on('on chat join', function(name) {
```

(continues on next page)

(continued from previous page)

```
                    if (choosing_name) {
                        $('#chat_form').attr("class", "chat_input");
                        $('#sendButton').text("Send");
                        $('#users_online_section').show();
                        $('#pic_frame').show();
                        $('#color_selection').show();
                        $('#username_disp').text(name);
                        choosing_name = false;
                    }
                });

                // Update the list of users.
                socket.on('userlist update', function(user_data) {
                    users = 0;

                    $('#users_online').empty();

                    // The server sends a list of user objects with the socket.id␣
↪excluded.
                    // These will be put into the list.
                    for (var i = 0; i < user_data.length; i++) {
                        var user = user_data[i];

                        if (user) { // check that it's not null
                            users++;

                            // insert the commas for the list
                            var text_format = user.username;
                            if (users > 1 && i < user_data.length) text_format = ", "␣
↪+ user.username;

                            // have the color of the users correspond with the user␣
↪color
                            var temp_span = $('<span>');
                            temp_span.attr("id", "user_" + user.username);
                            temp_span.text(text_format);
                            temp_span.css("color", user.user_color);

                            $('#users_online').append(temp_span);
                        }
                    }

                    // show number of users online
                    $('#users_online_title').text("Users Online (" + users + ")");
                });

                // Remove a user from the user list when they disconnect.
                socket.on('userlist remove', function(user) {
                    users--;
                    $('#users_online_title').text("Users Online (" + users + ")"); //␣
↪update this
                    // removes them from the list by id, id format is user_username␣
↪(ex: user_dog)
                    $('#user_' + user.username).remove();
                });

                // When someone changes their user color, update this in the user␣
↪list.
```

(continues on next page)

(continued from previous page)

```
                socket.on('userlist update color', function(user) {
                    $('#user_' + user.username).css("color", user.user_color);
                });
            });
        </script>
    </body>
</html>
```

The following is the server-side Node.js code.

Listing 26: Server Side Code for Public Pixel Art

```
var app = require('express')(); // Implement express module
var http = require('http').Server(app); // Implement HTTP module and use it with Express
var io = require('socket.io')(http); // Implement socket.io module and use it with HTTP
var port = 3000;
var users = {}; // The set of all users connected to the server.
// A 2d array of pixels for the picture. Format: picture[col][row] = pixel RGB color
var picture = [...Array(42)].map(e => Array(64).fill("rgb(255, 255, 255)")); // JavaScript does not have a nice way to make 2d arrays.
var name_dict = []; // Used as a hash map to check if there are duplicated user names.

// Discussed earlier, some routing through Express.
app.get('/', function(req, res) {
    res.sendFile(__dirname + '/index.html');
});

// Also discussed earlier, asynchronous listen that runs when port is established.
http.listen(port, function() {
    console.log('Server is running on port ' + port);
});

// Also asynchronous, runs when a user connects to the server.
io.on('connection', function(socket) {
    // Set up default user data for the user who connected to the server.
    users[socket.id] = {};
    users[socket.id].username = "Unconnected";
    users[socket.id].name_chosen = false;
    users[socket.id].user_color = "rgb(0, 0, 0)";

    // Tell the client (and only this client, that's why it's io.to) that they connected to the server.
    io.to(socket.id).emit('on page loaded', picture);

    // This runs when the user tries to change their user color.
    socket.on('update user color', function(color) {
        // If their request color is valid, set it to that, otherwise set it to a default color.
        var local_color = "rgb(0, 0, 0)";
        if (isValidColor(color)) local_color = color;

        // Don't bother changing it if it's the same color.
        if (local_color !== users[socket.id].user_color) {
            users[socket.id].user_color = local_color;
            io.emit('userlist update color', users[socket.id]); // update this color in user list as well
```

(continues on next page)

(continued from previous page)

```
        }
    });

    // Runs when a user sends a message.
    socket.on('chat message', function(msg){
        /* The input box for the chat and the username input are the same. So, when they send a message
            it needs to check whether they are sending a chat message or inputting their username.
        */
        if (users[socket.id].name_chosen) {
            // If they are sending a chat message, send it to all connected clients.
            io.emit('chat message', users[socket.id].username, msg, users[socket.id].user_color, "");
        } else {
            // If they are inputting their username, set up the following,

            // don't allow two users to have the same name
            var temp_name = msg.substring(0, 20); // limit usernames to 20 characters

            if (name_dict[temp_name] === undefined) {
                name_dict[temp_name] = 0;
            } else {
                // If two users have the same name, make the new user have a number after their name.
                name_dict[temp_name] = name_dict[temp_name] + 1;
                temp_name = msg.substring(0, 17) + name_dict[temp_name];
            }

            users[socket.id].username = temp_name; // set their username
            users[socket.id].name_chosen = true;

            // Send a message to all clients telling them that someone has joined the chat.
            io.emit('chat message', "", "", 'green', users[socket.id].username + " has connected to the chat.");
            io.to(socket.id).emit('on chat join', msg); // Tell the client that they set their username and have joined successfully.

            // Update the user list.
            var local_user_data = [];

            for (var temp in users) {
                local_user_data.push(users[temp]);
            }

            io.emit('userlist update', local_user_data);
        }
    });

    // Runs when a user clicks on a pixel.
    socket.on('box clicked', function(click_data) {
        // Only allow users who have selected a username to edit the picture.
        if (users[socket.id].name_chosen) {
            // Have default color be red, if the user has a valid selected color set it to that instead.
            var local_color = "rgb(200, 20, 20)";
```

(continues on next page)

(continued from previous page)

```
      if (isValidColor(click_data.color)) local_color = click_data.color;

      // Set the new color of the pixel for the client and server.
      picture[click_data.x][click_data.y] = local_color;
      io.emit('update box', click_data);
    }
  });

  // Runs when a user leaves the chat.
  socket.on('disconnect', function() {
    // Send a message to all clients that someone left.
    io.emit('chat message', "", "", 'red', users[socket.id].username + " has left␣
↪the chat.");
    io.emit('userlist remove', users[socket.id]); // remove them from the user␣
↪list
    users[socket.id] = null; // remove their information from the server
  });

  // This method uses regex to check that a color is in valid rgb format so that it␣
↪can be inserted into the
  // picture array without issues.
  function isValidColor(color_str) {
    var check = /^[r][g][b][(]\d{1,3}[,]\s?\d{1,3}[,]\s?\d{1,3}[)]$/;
    return check.test(color_str);
  }
});
```

3.4.6 Conclusion

In Conclusion, Node.js is a great tool for full-stack developers and for creating efficient, fast, and scalable web applications. Its support through modules such as Socket.io, Express, and more, allow you to easily implement features that would be complicated to create in other languages. Node.js is constantly being updated and is continuing to advance. Its asynchronous design prevents it from stalling or blocking. All around, Node.js is a great runtime environment if you want to quickly create a quality web application using JavaScript.

3.4.7 Citations

3.5 React JS

3.5.1 Introduction

ReactJS is a JavaScript library used for building user interfaces. Its goal is to make it easier to change an interface at any point in time by dividing the user interface into a group of components. A large reason it has become so popular is because of its higher efficiency and less complexity of other competitors such as Angular and Vue. It is also a project that benefits from being created and backed by Facebook. Due to React's combination of flexibility, ease of use, and efficiency, it is a highly used and demanded skill for jobs that work with modern web applications.

3.5.2 History

ReactJS originally started as a Javascript port of XHP, a version of PHP created by Facebook. The problem was to have dynamic web applications, it requires many trips to the server, which is not ideal for XHP. So, a Facebook engineer, Jordan Wilke, took it to the browser using Javascript; the result being ReactJS. [ReactHistory] The library was first created in 2011 and was first used in Facebook's newsfeed. Later, Instagram also implemented the library. It was open sourced in May of 2013. In 2015, React Native was introduced. This was to make for easier development with Android and iOS development. [Timeline] At first, people were unsure of React, but to combat this, they wanted to spread the message on how React is stable. This was done by having a "React Tour" to hopefully 'turn haters in advocates'. [Timeline] Today, React is mainstream and new versions are being released regularly

3.5.3 How React is Used

ReactJS works by storing the state of an application internally, then only re-rendering the content when the state changes. The largest piece of content in all React applications are called components. It renders some sort of output such as a button or input field. To write these components, a javascript function or class can be used. These components will correspond and change other interface elements. In the tutorial I have prepared, I will show how a simple form and button can be created using components and how these componenets can change interface elements. [FullStackReact] Another important aspect of React are States. These allow components to change the interface based on events such as a button click.

To be able to use ReactJS, we will use Javascript; more specifically, a React extension called Javascript eXtension, known as JSX. This extension allows us to write JavaScript that looks like HTML. To see this, we can look at Listing 1, a simple Hello World component:

Listing 27: ReactJS Helloworld

```
class HelloWorld extends React.Component {
  render() {
    return (
      <h1>Hello World</h1>
    );
  }
}
```

Observing the code, it appears as though the render() function is returning HTML, however it is JSX. At runtime, the JSX is then translated to regular Javascript:

Listing 28: Javascript Helloworld Translation

```
class HelloWorld extends React.Component {
  render() {
    return (
      React.createElement(
        'h1',
        'Hello World'
      )
    );
  }
}
```

[FullStackReact]

3.5.4 Tutorial

To begin building a React app using HTML, the source of React needs to be set inside a `<script>` tag inside the `<head>` element. This also includes a script that allows for the library, Babel. It is used to transpile ES6 JavaScript into ES5 JavaScript so the application can be compatible with more browsers. In this example, we will be building a simple form that asks for your name, then sends a simple welcome message based on your input. [FullStackReact]

Listing 29: ReactJS Libraries

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello World</title>
    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
  </head>
```

Now, looking at the body, before the Babel script we must set `<div>` tags to tell where the elements should render in the Document Object Model (DOM). Next, we can create our form component using a `constructor()`. Then, we set the initial state of the input value in line 10. We use `props` to allow for customization in case we need other forms. Also, in the constructor (Lines 12 and 13), we bind events to the component.

Listing 30: Initial Input State

```
1    <body>
2      <div id="root"></div>
3      <div id="welcome"></div>
4
5  <script type="text/babel">
6
7  class NameForm extends React.Component {
8    constructor(props) {
9      super(props);
10     this.state = {value: ''};
11
12     this.handleChange = this.handleChange.bind(this);
13     this.handleSubmit = this.handleSubmit.bind(this);
14   }
```

Next, we need methods to be able to handle events in the component such as button clicks or inputting a name. So, we have a method, `handleChange(event)` that will set the state value to the user's input. Then, another method, `handleSubmit(event)` that will be called in the event of the user clicking the Submit button. In the event of a submission, `ReactDOM.render()` will produce the `element` where the id, `"welcome"`, is found.

Listing 31: Handle Change

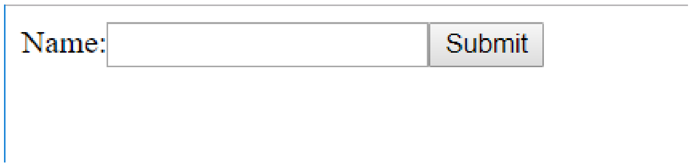
```
  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    const name = this.state.value;
    const element = <h1> Hello, {name}</h1>;

    ReactDOM.render(
      element,
      document.getElementById("welcome")
    );
    event.preventDefault();
  }
```

The `render()` is required for every React component that is created. In this instance, it creates the textbox for the user to input a name, then the Submit button. Finally, as seen in Listing 5, `ReactDOM.render()` is used to be able to call to the DOM. The function has two arguments, with the first telling the program *what* to render and the second *where*. In this case, we are rendering the `NameForm` component where the HTML element with an id, `"root"`, is located. [FullStackReact]

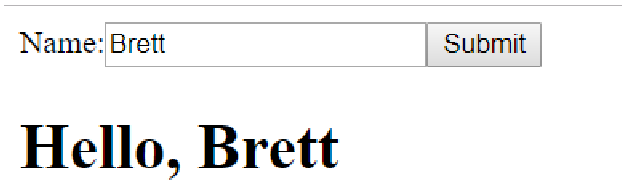
Listing 32: Render Components

```
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}

ReactDOM.render(
  <NameForm />,
  document.getElementById("root")
);


    </script>
</body>
</html>
```

If done correctly, you should see a very simple form with one a textbox and button:



After entering a name, by clicking the Submit button, it will call the `ReactDOM.render()` function that will render the element `<h1> Hello, {name}</h1>` where `{name}` changes based on the `state`. As previously mentioned, this is done where the `"welcome"` id is located. The page should look similar to this:



3.5.5 Advantages

One of the biggest advantages React has over other libraries is that is uses a Virtual DOM. So, instead of changing the document in the browser, it does these changes on a DOM that is run from memory. [Hackernoon] Using the Virtual DOM, React determines which components have changed and only sends those changes to the browser's DOM instead of reloading the entire page. This makes for a boost in performance, which of course is the goal for all businesses and companies that have an online presence. Reduced page load time will help with Search Engine Optimization and improve app's rankings on Google search. [Medium]

Another feature in React that helps with efficiency is its use of "Single way Data Flow." This means instead of the user interface element changing the model state, the model is updated first, then renders the user interface element. The changes are detected with a callback function, then those changes flow to the user interface. Using one-way data flow is easier to debug and more efficient than two-way data flow. [Neuhaus]

3.5.6 Disadvantages

Of course, there are always some disadvantages with any system. A couple commonly discussed downsides with React is its limitation of documentation. It hasn't been around as long as other libraries such as Angular, but Vue is newer and is already doing better in this aspect. React needs to figure out how to fix its lack of information on how to use and implement it. Another question surrounding React is its dependence on external libraries. Sometimes we see React depend on too many libraries, which could affect performance. [Medium]

3.5.7 Future of React

React's primary competitors in the library and framework market are Angular and Vue. The biggest difference between Angular and React is that Angular is more of a framework because of its structure. It is a "complete solution", meaning it is easier to start working instead of having to figure out libraries and packages. On the other hand, React and Vue are more flexible. Their libraries work with many different types of packages. There aren't many rules or guidance with these libraries, so it may be easier to run into problems than with Angular. However, out of the three, Angular has the steepest learning curve. The easy setup is beneficial, but it may be hard to understand what is going on within the pre-existing code. Another important note is that right now many believe Vue is the easiest to use because of the code readability and overall simplicity. [Neuhaus]

Putting all the advantages and disadvantages aside, React has beaten out its competitors in terms of market demand. As of June 2018, 28% of job postings have mentioned React while the next closest is Angular with 6.5%. React is also easily leading in the amount of NPM downloads at over 500 thousand compared to around 50 thousand. [Hackernoon]

3.5.8 Conclusion

React is a library that we are only getting started exploring and learning its capabilities. Its efficiency makes it desireable for companies. Simply put, the advantages out weigh the disadvantages. As proven by its market demand, it is a skill that is important to know for modern web application development and will not be going away in the foreseeable future.

3.5.9 Sources

3.6 Data-Driven Documents

D3.js is a JavaScript library that allows developers to easily manipulate documents based on data. It does this through a combination of HTML, CSS, and SVG creation and manipulation. The main point of D3, which stands for data driven documents, is providing a simple way to create powerful visualizations in a web page from data. It does not attempt to solve every problem with front-end development, but rather focuses on providing a solution for efficiently manipulating documents based on the data provided. D3 is extremely fast and is able to support large datasets being manipulated because it has little overhead. D3 allows developers to more easily integrate the use of data into their web page without hurting the performance of the page itself. This article describes the history of D3 and how it is used, as well as some examples of what can be done with the technology.

[Bostock1]

3.6.1 History

D3.js was initially released in 2011 by Michael Bostock as well as a number of his colleagues from the Stanford Visualization Group. It was created as a successor to the Protovis framework, which was also created by the same group. The main focus of D3 is to assist with data visualization on the web, and its goal is to provide the functionality of tools like excel, while also giving the efficiency of tools like OpenGL. It is still being maintained by Mike Bostock, and currently it has an open source BSD-license and is widely adopted and used at a professional level.

[Murray]

3.6.2 Using D3.js

It is very easy to implement D3.js into any web development project that you may be creating. All that needs to be done is include a link to the script in the bottom of your body tag in order to access all the functionality that D3 has to offer. The page will look like:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Title</title>
    </head>
    <body>
        <h1>Hello World!</h1>

        <script src="https://d3js.org/d3.v5.min.js"></script>
    </body>
</html>
```

Once you have added this to the body of your project, any script, internal or external, that is called and used within the webpage can use the d3 object to access and utilize the multitude of capabilities provided by the D3 library.

Selection

Selections is the main functionality that is provided by D3 that leads to a number of possibilities. These selections are used to simplify the process of manipulating elements within a web page. They can be created by using anything from element tags, class, id, or even containment, and is greatly simplified through the use of the D3 library. For example, if you wanted to set all of the header tags in a document to pink using the normal DOM, or Document Object Model, your code would look like the following:

Listing 33: DOM pink headers

```
var headers = document.getElementsByTagName("h1");
for(var i = 0; i < headers.length; i++) {
    var header = headers.item(i);
    header.style.setProperty("color", "pink", null);
}
```

However, using the D3 library, this can all be handled within one line of code:

Listing 34: D3 pink headers

```
d3.selectAll("p").style("color", "pink");
```

[Bostock2]

Dynamic Properties

Using selections to change the style is only an introduction to the capabilities of the D3 library. It can be further used to not just to change the style, but to change it dynamically and manipulate it with actual data you want to display on your web page. For example, say we have a basic web page that is using the D3 library with 6 `<div>` tags, each using the class "col-4", as shown below.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Selection</title>
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/
↪css/bootstrap.min.css" integrity="sha384-
↪Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin=
↪"anonymous">
    </head>
    <body>
        <div class="container-flex">
            <div class="row" style="height:300px">
                <div class="col-4"></div>
                <div class="col-4"></div>
                <div class="col-4"></div>
            </div>
            <div class="row" style="height:300px">
                <div class="col-4"></div>
                <div class="col-4"></div>
                <div class="col-4"></div>
            </div>
        </div>

        <script src="https://d3js.org/d3.v5.min.js"></script>
```

(continues on next page)

(continued from previous page)

```
    </body>
</html>
```

Once this page has been created it is easy to dynamically change the color of these boxes using the selections that are shown above. All that would be needed is a simple script being executed on the page similar to the one below:

```
<script>
    d3.selectAll(".col-4").style("background-color", function() {
        return "hsl(" + Math.random() * 360 + ",100%,50%)";
    });
</script>
```

Likewise, to label each of the boxes with their corresponding number, you would begin to add data as part of your selections and inject it onto the page.

```
<script>
    d3.selectAll(".col-4").data([1, 2, 3, 4, 5, 6]).append("h1").text(function(d) {
        return "Box " + d;
    });
</script>
```

What this code does is create a set of data that becomes associated with the selection made. Once the original selection is made, it injects an `<h1>` tag within each of the `<div>` tags that are selected. Then, using the `.text()` function, it will manipulate the text contained within the `<h1>` tag using the data being passed into the function. When all of this code is put together, we end up with a web page that looks like the following.

dynamic-properties-josh

[Bostock1]

Interactive SVGs

Another benefit that D3 provides is that ability to create and manipulate SVGs in real time. Not only this, but the SVGs can also be interactive with the user. To do this, it uses the same selection and injection tools that have been used in previous examples, but it uses them in a different way. In this process, an `<svg>` is created to house the graphic, and then a `<rect>` is injected inside of that to give the SVG a specific size. To create an interactive SVG, you will also add a function call to `.on()` that will check for movement within the `<rect>` and call the `particle()` function when there is, as seen below.

Listing 35: Creating an interactive SVG

```
var width = innerWidth, height = 500;

var svg = d3.select("#interactive-svg").append("svg")
    .attr("width", width)           // Setting attributes of the SVG
    .attr("height", height);

svg.append("rect")
    .attr("width", width)
    .attr("height", height)
    .on("ontouchstart" in document ? "touchmove" : "mousemove", particle); // On␣
↪cursor move with ternary if/else statement
```

Once the `<rect>` has been created, then the `particle()` function must be created. The function will use the method `d3.mouse(this)` to determine the location of the cursor at that moment. Once it has the location, it creates a `<circle>` tag within the `<rect>` and places the center at the location of the cursor. Once you have done this, you set the color of the circle, and then call the `.transition()` function. This will begin an animation of the circle, but it needs other data to determine how to transition. To begin, you set the duration of the transition with the `.duration(time)` method. Then you set the easing of the transition with the `.ease(speed)` method. Finally, you set the final attributes that you want the object, in this case a circle, to have, and then remove it with the `.remove()` method.

Listing 36: Dynamically creating circles

```
function particle() {
    var m = d3.mouse(this);

    svg.insert("circle", "rect")
        .attr("cx", m[0])
        .attr("cy", m[1])
        .attr("r", 1e-6)
        .style("stroke", function () {
            return "hsl(" + Math.random() * 360 + ",100%,50%)";
        })
        .style("stroke-opacity", 1)
        .transition()
        .duration(2000)
        .ease(Math.sqrt)
        .attr("r", 100)
        .style("stroke-opacity", 1e-6)
        .remove();

    d3.event.preventDefault();
}
```

Shown below is this interactive SVG in action.

[Bostock3]

3.6.3 Who uses D3?

Since D3 is a JavaScript library designed specifically for simpler creation and manipulation of graphics using data, it is not as popular as some of the other JavaScript libraries and frameworks. However, it is still used by a number of professional organizations to graphically display data to customers and users. This is often achieved through a user dashboard or a data analytics tool that is part of the application. Some of the better known organizations that utilize the D3 library are given below.

- Acorns
- 23andMe
- Square
- Coursera
- Free Code Camp
- Weebly

[StackShare]

3.6.4 Conclusion

D3.js is a very beneficial library when it comes to data display and manipulation on web pages. It also allows for the injection of dynamic graphics and properties that allow you to greatly improve the interface of your application. It can greatly increase the effectiveness and general look of any data analytics tool and opens up a number of possibilities in regards to graphics. It was been widely accepted and implemented in professional development society, and allows for the efficient manipulation of web pages while still creating a friendly user interface.

3.6.5 Sources

3.7 NodeJS

3.7.1 Introduction

Node.js runs on a cross-platform of JavaScript's runtime environment. It was made to fix many problems like platforms and the performance in network communication time, and helps to reduce the time it takes to process web requests and responses. Node.js runs the V8 JavaScript engine which can leverage the work of engineers which makes this software perfect for real-time applications that run across distributed devices. This examination of Node.js highlights the details about Node.js and why you should use it and sample code to help you better understand why Node.js is growing in popularity. [IntroNodeJS]

3.7.2 History of topic

Node JS was made in 2010, which makes it only 10 years old. JavaScript, on the other hand, is 24 years old. Node.js was written by Ryan Dahl and other developers working at Joyent, a software company that specializes in application virtualization and cloud computing, in 2009. The first release of Node.js only supported Linux and Mac OS. Two years later the Node Package Manager, or NPM, was released which allowed for the sharing of open source libraries. Dahl was not happy with the way Apache HTTP servers handled concurrent connections and the way code was being created. This inspired him to create the Node.js project which he publicly showed at the inaugural European JSConf on November 8, 2009. This showing gave him and Node.js a lot of publicity and won him a standing ovation. [IntroNodeJS]

In June 2011, Microsoft worked with Joyent to implement a Windows version of Node.js. In 2012, Dahl stepped aside and promoted the creator of NPM Isaac Schlueter to take over the project. Two years later, Fedor Industry started a fork of Node.js called io.js. This caused much conflict at Joyent so a neutral Node.js foundation was created. In June 2015, the Node.js and io.js communities decided to work together under this newly formed Node.js foundation. [IntroNodeJS]

3.7.3 What is nodeJS

Based on the official Node.js documentation, Node.js is defined as "a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices." This server-side platform is built on Google Chrome's JavaScript V8 Engine. Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Most Node.js applications are written in JavaScript but can run on multiple operating systems like Windows, Linux, and X. [Node.jsIntroduction]

3.7.4 Features

There are many features and reasons that software architects should make Node.js their first choice of software to use. Being built on Google Chrome's V8 JavaScript Engine makes the library and code execute very fast. Its I/O is Asynchronous and Event Driven which makes the APIs of Node.js all asynchronous and non-blocking. This makes it so a Node.js server will never have to wait for an API to return data. Since Node.js has the event mechanism that helps the server respond in a non-blocking way it makes Node.js very scalable. [WhyUseNodeJS]

Along with these features of Node.js, it is also single threaded, has no buffering, and is open source. All of these are great features that give Node.js the leg up against its competitors and ultimately is a top choice for this kind of software. [Node.jsIntroduction]

3.7.5 Who uses it

Node.js is used by some of the largest corporations in the world. It is used by applications and businesses that you use in your everyday life like Netflix, Walmart, Microsoft, Uber, PayPal, LinkedIn, EBAY, NASA, and much more.

3.7.6 Advantages and disadvantages

As stated before, there are many benefits to using Node.js opposed to the other popular server-side programming languages. Node.js offers easy scalability. Applications can be scaled easily in horizontal and vertical directions and makes it very easy to add nodes to the existing system. Node.js is also very easy to learn. Since JavaScript is such a popular programming language, most developers already have knowledge on it and makes it much easier to start using Node.js at the back-end. This saves a lot of time learning how to use Node.js because most people will already have experience with JavaScript. [NodeJSWebApp]

Node.js includes other benefits like Full stack JS, which offers a high performance, support from a large and active community, caching, freedom to develop apps, commonly used tools, handles requests simultaneously, and is highly extensible.

These are just some of the reasons that make Node.js stand out to its competition, but not every aspect of Node.js is a positive. One of the main problems that developers face is that the Application Programming Interface (API) keeps on changing and is not stable. This can result in the developers being forced to make changes to accessible code bases to match the latest version of the Node.js API which is a waste of time and very repetitive and inefficient. Node.js also does not have a strong library support system. Node.js has also adapted an asynchronous programming model. With there being more advantages for developers in comparison to other languages, Node.js is being adopted by more and more business organizations and is gaining extreme popularity. [NodeJSWebApp]

3.7.7 Before NodeJS

Before Node.js, the way JavaScript ran on servers was very inefficient. Users would have to use Netscape's LiveWire Server or Microsoft's Active Server Pages (ASP). If they were not using one of these two then they would have to use other third-party server products that supported JavaScript. [BeforeNodeJS]

Microsoft's software started dying out in 2002, when Microsoft replaced ASP with ASP.NET. This replacement software favored using C# instead of JavaScript which made it lose popularity at a very rapid rate. Before Node.js, JavaScript never communicated with the database. The only option at this time was to have a backend language like PHP, ASP, JSP, and others retrieve data from the database and send the data to JavaScript. [BeforeNodeJS]

3.7.8 How it works

Node.js operates asynchronously and uses event-loop mechanisms to function. If you look at the example below, you will see that when socket.listen(4000) executes, a Web-Socket server is created on a single thread event loop which listens continuously on port 4000 until told otherwise. When you connect to the server, the program runs the “onConnection” event which the loop picks up and publishes the data to the thread pool. This is the main difference between Node.js and other servers. Other servers have to create a new thread every time you want to connect to a server. With Node.js, it receives all the requests on a single thread and then delegates them to be handled by background workers. [SingleThreadMechanism]

3.7.9 Single thread mechanism code example

Listing 37: Single Thread Mechanism Code

```
var sockets = require('websocket.io'), httpServer = sockets.listen(4000);
httpServer.on('onConnection', function (socket) {
console.log('connected......');
httpServer.send('Web socket connected.');
httpServer.on('message', function (data) {
console.log('message received:', data);
});
httpServer.on('close', function () {
console.log('socket closed!');
});
});
```

3.7.10 What makes it unique

Node.js has a unique advantage compared to its competitors. Millions of frontend developers that write JavaScript for the browser are not able to write the server-side code and the client-side code without needing to learn and implement a different programing language or software. Node.js is also able to handle thousands of connections with a single server without having to manage thread concurrency. This is significantly more efficient and reduces a large number of bugs that would occur if managing thread concurrency was implemented. [IntroNodeJS]

3.7.11 Sample code

Listing 38: Example Code

```
//server.js
    const http = require('http'),
          url = require('url'),

    makeServer = function (request,response){
       let path = url.parse(request.url).pathname;
       console.log(path);

       if(path === '/'){
          response.writeHead(200,{'Content-Type':'text/plain'});
          response.write('Hello world');
       }
       else if(path === '/about'){
         response.writeHead(200,{'Content-Type':'text/plain'});
         response.write('About page');
       }
       else if(path === '/blog'){
         response.writeHead(200,{'Content-Type':'text/plain'});
```

(continues on next page)

(continued from previous page)

```
        response.write('Blog page');
      }
      else{
        response.writeHead(404,{'Content-Type':'text/plain'});
        response.write('Error page');
      }
      response.end();
    },
    server = http.createServer(makeServer);
    server.listen(3000,()=>{
     console.log('Node server created at port 3000');
    });
```

As you can see in the example above, this is a simple example of Node.js code. If you go to "localhost:3000" and then go to "localhost:3000/about" or any of the other examples above, it will take you to separate pages with different messages. If you do something like "localhost:3000/PageDoesNotExist" it will give you an error page because we did not make this above in the code. This makes it so we can easily start a server, but this is inefficient to do every time you need a new web page on your server. This is just a simple example of how to get things started. [NodeJSTutorials]

3.7.12 Conclusion

Node.js has transformed the usability of JavaScript, making Node.js a complete and efficient programming language. Its I/O is Asynchronous and Event Driven which makes the APIs of Node.js all asynchronous and non-blocking and increases its overall efficiency. With all the advantages that Node.js brings to programming, its obvious to see why many large corporations take advantage of its benefits. All things considered, Node.js is an amazing open source, cross-platform runtime environment that excels at developing server-side and networking applications and continues to show why it is so efficient and popular in so many real world scenarios.

3.7.13 Sources

3.8 Responsive Web

When it comes to styling a web page or a mobile app or even being able to print out a web page, there are many tools that you can use. The main idea here is the CSS or a cascading style sheet. This special document usually holds all of the styling and formatting for the web page or mobile app you are designing. This could include anything that applies to the presentation of the content on the page; layout, colors, and even fonts. With a CSS file, it is easier to change the presentation of the content and allows the programmer to quickly change multiple aspects of the web page or app at once.

3.8.1 History of topic / library of code

There was not always CSS and the ability to change the styling and other aspects of the web browser. In the early days of web browsing, when Mosaic and and other early stage web clients were out, the only things you could change were the color and style of the font you see. Eventually, people complained about wanting custom touches and that they wanted more from their browsing clients. That is when a few people teamed up and made CSS. The first CSS idea was brought to attention by Bert Bos and Håkon Wium Lie. [CSSHistory]

3.8.2 Responsive Web Design

- We know that CSS can change the look or application to different things within a webpage or app, but what can it change or do?
- The CSS can change many things very simply. A few of the easier things that you can change in the CSS are the colors and the fonts.
- The background color is easily changeable with a simple selector tag and a few adjustments. Here is how the selector tags are done:

Listing 39: Selector Example

```
#p1 {background-color:rgba(255,0,0,1);}
#p2 {background-color:rgba(0,255,0,1);}
```

- Here is how the selector tags are applied to paragraph tags:

Listing 40: Applying Selectors

```
<p id="p1">Red</p>
<p id="p2">Green</p>
```

- Once these tags are applied to the <p> tags above, the background of those words will appear red and green like this:

[w3SchoolsRef]

- You can also change the colors using hexadecimal color codes, something like this:

Listing 41: Hexadecimal Color Code Changes

```
#p3 {background-color: #FF0000;opacity:1;}
<p id="p3">Red</p>
```

This will do the same thing as above however using that special code. You can find these codes at this website, Hexadecimal Codes.

When it comes to resizing images and text on shrinking and growing screen sizes you can put responsive images and text in so it always looks proportional. The way to do it for an image is quite easy. All you have to do is add in the max-width style and it will not get bigger than its original size but will shrink to smaller screens as well!

- Here is how you do it:

Listing 42: Responsive images and text

```
/* This is a responsive Image */
<img src="picture.PNG" style="max-width:100%;height:auto;">

/* This is responsive Text */
<p style="font-size:10vw">Responsive Text!!!</p>
```

3.8.3 Media Queries

So, the first thing you need to do for a webpage would be add the <meta> tag in all your web pages. This allows the page, text, images and much more to shrink and grow appropriately with the page that you are looking at.

Listing 43: viewport

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

When it comes to Media Queries this is where format of the web page really comes into play. The media query is a rule that uses the identifier @media and only applies the CSS to a code block if a certain condition is true.

One quick example would be to make the background of the <b> (body) tag a different color using the @media selector. This is easy, all you have to do is set what you want to happen with a condition. So something like this:

Listing 44: @media example

```
@media only screen and (max-width: 600px) {
  body {
      background-color: blue;
  }
}
```

So in this example, if the screen was 600 pixels or smaller then the background of the body would change to blue from whatever it was before.

How to change the font size of your text based on screen size:

Listing 45: Change in font size

```
@media only screen and (min-width: 601px){
      div.whateverTextYouWantToChange {
            font-size: 80px;
      }
}

@media only screen and (max-width: 600px){
      div.whateverTextYouWantToChange {
            font-size: 30px;
      }
}
```

Another cool thing you can do is hide images. If the screen is too small to view them or you do not want a smaller device to render in a large picture, for example.

Listing 46: How to make an item disappear! (TA-DA)

```
@media only screen and (max-width: 600px){
      div.itemNotShown {
            display: none;
      }
}
```

One thing that was mentioned on many of the sites I looked through, was you should always code for the smaller screens first. Scale up rather than down. So for example, instead of saying if the screen gets too small then change it. Make it so if it gets too big then change it. This way your website or app will load faster on the smaller screens.

The only thing that needs to be changed in the CSS when designing for mobile devices first is making it so instead of shrinking to size, we are growing. So, when making our columns for a page we will usually make it so each column takes up 100% of the width of the screen. This will allow cellphones to load faster and if the page gets larger than a certain size, then we change to columns taking up a certain percentage of the screen. For example.

Listing 47: Mobile Devices First!

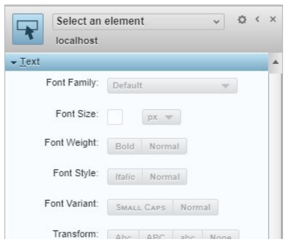
```css
/* This is for the cellphones, it makes the columns 100% width of the screen and the columns stack*/
[class*="col-"] {
      width: 100%;
}

/* Then if we hit 768px or greater we switch to columns taking up a percentage*/
/* of the screen and they are no longer stacking*/
@media only screen and (min-width: 768px) {
      /* this column will take up 25% of the screen, if assigned to a tag*/
      .col-1 {width: 25%;}
      /* this will take up 50% of the screen */
      .col-2 {width: 50%;}
}
```

[MediaQueries] [w3SchoolsMediaQueries]

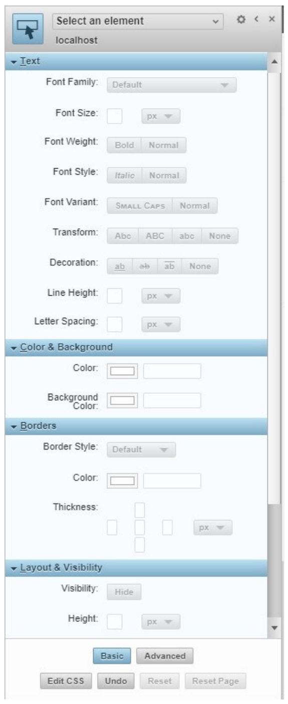
3.8.4 Stylebot

Style bot is an incredible tool for programmers to help understand and better their CSS code and writing ability. This tool works in sync with the Chrome web client and allows the user to change the CSS to the page on the fly. This will help the programmer or user to better understand what is going on, and give them a preview of what they changed almost instantly.

The way style bot works is you open it up and it will appear on the side of your Chrome client. In the side panel, where style bot is, you will see most of the selectors or things you can change on the website you are looking at.



Once you have chosen one of these options, you can choose what to do with it! There will be many options from font size, font style, font family, underline, letter spacing, color of the letters, background colors and much more. Here you can click what to apply and the CSS will automatically be shown on your instance of the website.



This is a great tool for changing your already built website's CSS to see if any improvements can be made to the style or format. Also, it is useful for personal use. If you are colorblind for example you can change the colors on a website to make it more user-friendly for yourself. Once you are done with the CSS options you can see the CSS code you changed! [diviSpace]

3.8.5 Print CSS

When it comes to CSS and printing paper there are a few things to take into consideration. You want to be able to have both a screen or online CSS but also you want a printing CSS. This will allow the user to apply the correct style sheet when it comes to printing or displaying the page correctly. Here are a few questions you should ask about your website before making a print CSS:

- Is there clutter on the screen, when printing?
- Is there printing cost limitations involved?
- What is not needed on a printed piece of paper?

Like the pesky navigation menu that looks like this:

- Index
- Tab 1
- Tab 2
- Tab 3

This is how to get rid of it, just like the example above when not displaying images:

Listing 48: Getting rid of NAV menu

```
/* how to get rid of the nav*/
header nav {
```

(continues on next page)

(continued from previous page)

```
      display: none;
}
```

Another unnecessary thing would be most all media options, like a video for example. Why would you need a video on a piece of paper? This is how you would take it out:

Listing 49: Getting rid of NAV menu

```
header nav, video, audio {
      display: none;
}
```

Another good idea would be making your images not as big or scale to the page so they do not go over the edge of the page. This can be done with the max/min-width tag like mentioned before, or you can set the images to a specific size.

Listing 50: Resizing images

```
img {
      max-width: 500px;
}

/* OR */

img {
      max-width: 100%;
}
```

Another thing you can do is change your fonts or size of fonts to your liking, depending on the different columns or text blocks.

Listing 51: Resizing images

```
/* This will change your first headers font size*/
h1 {
      font-size: 24pt; /* Change to any size you would like */
}

/* this will change your font for the body tag */
body {
      font: 12pt “Times New Roman”, serif;
}
```

All of these things in the end will make your website look better on print as well as save ink and paper. Whether that is the ability to change font size or make an image disappear either way there are many things you can do with CSS and printing but these were some of the basic things you can do with a print CSS. [SmashingMagazine]

3.8.6 Conclusion

Overall, when it comes to CSS on your website or mobile app all of these tools above can be extremely helpful. For formatting, styling, and perfecting a website or app CSS is needed and most likely will stay that way for awhile, until something better comes out. More tools will come out to make CSS easier to implement like Style bot but the core of CSS will remain. CSS is embedded into pretty much every website you can think of when it comes to online and without it everything would look bland, boring and just not appealing.

Sources

3.9 Google Accelerated Mobile Pages

The **Accelerated Mobile Pages** (AMP) Project, is an open-source HTML framework created by Google, used to create web pages that load smoothly and quickly. AMP prioritizes end user experience, even if it is harder on the developer. It is designed to “fix the web of today, not the web of tomorrow”. [AMP]. This means that when developers find optimizations that aren’t possible with today’s platforms, they should participate in the development of standards to get these optimizations implemented. Including components on your web page that can’t reliably load quickly, or perform at 60fps or higher, violates the reason that AMP exists, to make mobile pages load faster. This document will explain how AMP improves mobile performance, and why it is important to use.

3.9.1 History

Google announce the AMP project on October 7, 2015. Their goal was to create a tool to improve the performance of the mobile web. Over 30 news sites and technology companies were announced as collaborators, including Twitter, Pinterest, LinkedIn, and WordPress. The first AMP pages seen by the public were viewed in February 2016 when Google began showing AMP versions of web pages in its search results. Initially, AMP pages were only used for the “Top Stories” section of Google’s mobile search results. By September 2016, Google expanded this to the main search results. AMP links are designated with a lightning bolt symbol. (See example in Code Examples section)

In September 2016, Microsoft announced AMP support in Bing apps for mobile phones. A year after AMP was launched, Adobe reported that AMP pages accounted for 7% of all web traffic for top US publishers. By May 2017 Google reported that over two billion AMP pages had been published globally.

[AMPWiki].

3.9.2 How It Works

Optimization 1

AMP has 7 optimizations it attributes to its success in loading mobile pages. The first of these is to execute all JavaScript asynchronously. JavaScript is powerful, but can cause delays to a page’s rendering. To combat this, AMP only allows asynchronous JS(JavaScript), and AMP pages cannot include any author written JS. Instead of writing your own JS, interactive page features are handled by custom AMP elements. These elements might run JS themselves, but it has been designed to not cause performance issues.

Optimization 2

The next optimization is to size all resources statically. Any external resource like images and ads, must state their size in the HTML so AMP can layout each element’s size and position before anything else is loaded. This prevents pages from jumping around and changing layouts after these resources are loaded.

Optimization 3

The third optimization is not letting extension mechanisms block rendering. AMP supports extensions for things like Instagram embeds and tweets. While these require additional HTTP requests, these requests don’t block page layout and rendering. Any page that uses a custom script must tell the AMP system that they will eventually add in a custom tag.

Optimization 4

The fourth optimization is to keep all third-party JavaScript out of the critical path. In most cases, third party JS uses synchronous JS loading. For example, if you have five ads on your page, and each of them cause three synchronous loads, each with a 1 second latency, that is 15 seconds of loading just for JS. AMP pages allow third party JS but only in certain iframes. By only allowing them in iframes, they can't block the execution of the main page.

Optimization 5

The fifth optimization is that all CSS(Cascading Style Sheets) must be inline and size bound. CSS blocks all rendering, this causes the page load to get bloated. In AMP pages, only inline styles are allowed. This helps reduce the number of HTTP requests. The inline style sheet has a max size of 50kb, which is big enough for good looking pages, but still requires practice to keep things clean.

Optimization 6

The sixth optimization is to only run GPU(Graphics Processing Unit) accelerated animations. The best way to run fast optimizations, is to run them on the GPU. The GPU knows how to do different animations quickly, but it can't update the page layout. AMP only allows animating and transitioning with transforms and opacity so that the page layout doesn't need to be reloaded.

Optimization 7

The seventh optimizations is to prioritize resource loading. AMP controls the downloads for all resources and it loads only what is needed. When AMP downloads resources, it optimizes them so the most important resources are loaded first. Images and ads are only downloaded if they might be seen by the user.

3.9.3 Code Examples

Listing 52: AMP Hello World

```
<!doctype html>
<!-- This is the AMP declaration. `<html amp>` works too.-->
<html ⚡>

<head>
    <meta charset="utf-8">
    <title> Hello World</title>
    <!-- The AMP runtime must be loaded as the second
    child of the `<head>` tag.-->
    <script async src="https://cdn.ampproject.org/v0.js"></script>
    <!--
        AMP HTML files require a link pointing to the regular HTML. If no HTML
        version exists, it should point to itself.
    -->
    <link rel="canonical" href="https://2019-spring-web-dev.readthedocs.io/en/latest/
↪final/knouse/index.html">
    <!--AMP HTML files require a viewport declaration.-->
    <meta name="viewport" content="width=device-width,minimum-scale=1,initial-scale=1
↪">
    <!--CSS must be embedded inline.-->
    <style amp-custom>
```

(continues on next page)

(continued from previous page)

```
        h1 {
            color: black;
        }
    </style>
    <!--The AMP boilerplate.-->
    <style amp-boilerplate>body{-webkit-animation:-amp-start 8s steps(1,end) 0s 1
    normal both;-moz-animation:-amp-start 8s steps(1,end) 0s 1 normal both;-ms-
↪animation:-amp-start 8s
    steps(1,end) 0s 1 normal both;animation:-amp-start 8s steps(1,end) 0s 1 normal␣
↪both}@-webkit-keyframes
    -amp-start{from{visibility:hidden}to{visibility:visible}}@-moz-keyframes
    -amp-start{from{visibility:hidden}to{visibility:visible}}@-ms-keyframes
    -amp-start{from{visibility:hidden}to{visibility:visible}}@-o-keyframes
    -amp-start{from{visibility:hidden}to{visibility:visible}}@keyframes
    -amp-start{from{visibility:hidden}to{visibility:visible}}</style>
    <noscript><style amp-boilerplate>body{-webkit-animation:none;-moz-animation:none;
    -ms-animation:none;animation:none}</style></noscript>
</head>

<body>
  <!--
    Most HTML tags can be used directly in AMP HTML.
  -->
  <h1>Hello World!</h1>
  <!--
    Certain tags, such as the `<img>` tag, are replaced with equivalent or
    slightly enhanced custom AMP HTML tags
  -->
  <amp-img src="/static/samples/img/amp.jpg" width="270" height="150" layout=
↪"responsive"></amp-img>

</body>
</html>
```

The code above is a basic example for how to show text and images using AMP. As you can see, AMP is very similar to HTML in its use of tags. For anyone proficient in HTML, AMP would be fairly easy to pick up.

This next example shows a basic animation based on the scroll of the page:

Listing 53: Scrollbound Animation

```
<!doctype html>
<html >

<head>
    <meta charset="utf-8">
    <title>Scrollbound Animation Basics</title>
    <script async src="https://cdn.ampproject.org/v0.js"></script>

    <script async custom-element="amp-position-observer" src="https://cdn.ampproject.
↪org/v0/amp-position-observer-0.1.js"></script>

    <script async custom-element="amp-animation" src="https://cdn.ampproject.org/v0/
↪amp-animation-0.1.js"></script>

    <script async custom-element="amp-fit-text" src="https://cdn.ampproject.org/v0/
↪amp-fit-text-0.1.js"></script>
```

(continues on next page)

(continued from previous page)

```
    <script async custom-element="amp-carousel" src="https://cdn.ampproject.org/v0/
↪amp-carousel-0.1.js"></script>


    <link rel="canonical" href="https://2019-spring-web-dev.readthedocs.io/en/latest/
↪final/knouse/index.html">
    <meta name="viewport" content="width=device-width,minimum-scale=1,initial-scale=1
↪">
    <style amp-boilerplate>body{-webkit-animation:-amp-start 8s steps(1,end) 0s 1␣
↪normal both;
        -moz-animation:-amp-start 8s steps(1,end) 0s 1 normal both;-ms-animation:-amp-
↪start 8s steps(1,end) 0s 1 normal both;
        animation:-amp-start 8s steps(1,end) 0s 1 normal both}@-webkit-keyframes
        -amp-start{from{visibility:hidden}to{visibility:visible}}@-moz-keyframes
        -amp-start{from{visibility:hidden}to{visibility:visible}}@-ms-keyframes
        -amp-start{from{visibility:hidden}to{visibility:visible}}@-o-keyframes
        -amp-start{from{visibility:hidden}to{visibility:visible}}@keyframes
        -amp-start{from{visibility:hidden}to{visibility:visible}}</style><noscript>
    <style amp-boilerplate>body{-webkit-animation:none;-moz-animation:none;
        -ms-animation:none;animation:none}</style></noscript>

    <style amp-custom>

        .fidget-spinner-scene {
            margin: 10px 20%;
        }
    </style>
</head>

<body>
    <div class="fidget-spinner-scene">

        <amp-position-observer on="scroll:spinAnimation.seekTo(percent=event.percent)
↪" intersection-ratios="1" layout="nodisplay">
        </amp-position-observer>

        <amp-img id="fidgetSpinnerImage" width="256" height="280" layout="responsive"
             src="https://amp.dev/static/samples/img/fidget.png"
             alt="Picture of a fidget spinner"></amp-img>
    </div>

    <amp-animation id="spinAnimation" layout="nodisplay">
        <script type="application/json">
            {
                "duration": "1",
                "fill": "both",
                "direction": "reverse",
                "animations": [
                    {
                        "selector": "#fidgetSpinnerImage",
                        "keyframes": [
                            { "transform": "rotate(360deg)" }
                        ]
                    }
                ]
            }
        </script>
```

(continues on next page)

(continued from previous page)

```
    </amp-animation>
</body>

</html>
```

The example above works by rotating the image based on the users current position on the page. the “amp-position-observer” monitors the position of the element within the view of the user. The image rotates a full 360 degrees as the user scrolls by with { “transform”: “rotate(360deg)” }. There are many scrollbound animations you can do, including parallax scrolling as well as fades and slide transitions.

3.9.4 Criticisms

AMP does not have numerous criticisms, however, it has one big criticism that most of the tech industry agrees with; Google attempting to assert dominance on the web. Many people think AMP is an attempt for Google to dictate how websites are monetized and built. AMP has also been linked with Google’s attempts to deprecate URLs so users can’t immediately see if they are viewing a normal web page, or an AMP page hosted by Google.

Many people fear that even though AMP is open-source, Google is using it to shape how mobile web operates. It also ensures Google gets plenty of ad revenue from AMP pages. Google disagrees with this however, saying that AMP is an open-source initiative, and isn’t “Google’s AMP”.

Some publishers report reduced ad revenue when using AMP pages. Since AMP uses mainly banner ads, this prevents publishers from using pop-ups, or other customized ad placements. Some publishers defend AMP, saying that monetization has remained largely unchanged since using AMP. AMP has since launched the “AMP Ads Initiative” which supports more types of advertising options.

[AMPWiki].

3.9.5 Conclusion

AMP is a very useful tool for web developers of all skill levels. Websites created using AMP put the user first, by creating pages that load quick and don’t jump around when new ads or pictures load in. The optimizations that the AMP team has put in place really speed up the loading process for all devices, not just mobile devices. By using AMP, developers can create stunning pages with only a bit of practice and a bit of creativity.

3.9.6 Sources

3.10 Playing with ReactJS

3.10.1 Introduction

Javascript libraries and HTML coding has grown since the initial years of the world wide internet. Companies had abided with using HTML for years as it works and can be easily taught to people. One of the javascript libraries that exist is ReactJS, a free to use library developed by Facebook and released freely to anyone. ReactJS, despite being free, must be looked into detail before used, to understand what it is and how to use it.

3.10.2 ReactJS, What is it?

ReactJS or [rjs] is a Javascript library geared towards designing and updating user interface on websites while removing the requirement to update already existing code. The library is meant to handle the front-end development for user friendly interfaces in a constant change of our world and the need to get up to date information on a website. The started implimentation of ReactJS was for one website, facebook, and recently changed from a company exclusive to an open source resourcewhich people have contributed over the years. It is still used today by plenty of websites for its ease of implimentation since its release.

3.10.3 Reactjs brief history.

The way ReactJS had started out to achieve its user-friendly ease of use was through slow implimentation, taking two years according to RisingStack [rjs_history]. The original purpose of ReactJS was the need for a better code. This was a result of Facebook having increased amount of added app features around 2010 rendered the app a update logistical nightmare. The issue hit its peak when engineers at Facebook could not keep up with the updates. At the third year, ReactJS became open source with initial rocky starts but it has been able to continue to today's world. Since its free release, more and more sites have used Reactjs to improve their sites' user experiences but that does not mean Reactjs will be used forever as our technology develops.

3.10.4 ReactJS, who uses it?

Since Reactjs became open source, many websites have implimented ReactJS and some of the [built_sites] people use to this day. Its focus on a better user experience has drawn more and more users to sites using Reactjs with their code. One of the websites using this library, Imgur, an image sharing website, had increased its userbase significantly using ReactJS. Reactjs has shown to be a very effective tool for website development.

3.10.5 Reactjs with other libraries

Within web development there are times where sites do not exclusively use one library but multiple libraries. Reactjs seems to be one of the external libraries added onto existing code; which was the design purpose anyway. Reactjs is not what is cookie cut javascript library nor a framework, but both. Why [reactjs_classification] could be considered both is its flexibility and traits that define a library and framework.

3.10.6 Example/Experiment of Reactjs

The code above is a simple program that flips between a '-' and '|' on each mouse click. The squares will not return to a blank square. How it is made is using two classes, one for displayiny the boxes and another to handle changing the boxes when clicked. The box has functions to handle how many boxes are on screen, when the user clicks on a box, changing what the box contains, and swapping between '-' and '|' on each click. It is a small program which can be entertaining for a little while.

To break it down, this section of code above is the main display class. It simply calls the other class to output the boxes and their states constantly. It automatically updates itself while in use.

This section of code above is for the initial array that is used by the squares. It fills the squares with an empty null value that the user can see is empty before they modify it by clicking on the boxes.

The above code block, while it is quite large to interpet simply handle the squares. The first two functions, square and render square, handles the interaction of the squares when the user clicks on one of the squares. The last function, render, handles the status of the squares and keeps them in a standard array.

This section of code handles the data which will update the previous two square functions for changing between a - and a | to be displayed in each square. This does not allow a null to be entered again. The only way to get blank squares again is refreshing the page.

3.10.7 Reactjs Issues

There are some problems with Reactjs in comparing other libraries. Reactjs [versus] other libraries and reasons to [use_reactjs] and [not_use_reactjs] explain what those problems are. One of the key arguments is while it can be very flexible, it has a harder time being flexible in complex interactive web projects.

3.10.8 Additional Problems by Code Design

When the code was developed, there was difficulty finding an option to allow additional symbols to be possible when the user clicks. Research needs to be done with testing to see if the program could have something added to cycle between any number of symbols and letters even. The lack of research and time left to develop the code further limited what it could do significantly.

3.10.9 Conclusion

Reactjs is another library that has options. Some of these options are from its intended use before it became free to use by anyone. Reactjs does not seem like a final soultion to all the problems coding in HTML can bring but it has been used successfully on multiple websites. Reactjs is a quality optional library to use for projects within reason.

3.10.10 Sources

3.11 Vue- KDL

3.11.1 Why use Vue.js and Installation

Vue.js is a progressive framework built in Javascript that can build single paged application. It is also versatile and can be integrated into existing web pages. One of the biggest advantages of Vue is the ability to integrate and embed it into existing web projects. It is used only for front end development and is easy to learn using existing syntax pre-ES2015/ES6 standards. Vue.js is a great framework for people wanting to learn front-end development and for advanced users. [WhatIsVue]

The person behind the initial development behind Vue.js was Evan You. Evan originally worked at Google building web applications. He especially worked with the Angular front-end framework. He liked the framework so much, but he always felt like it was too resource heavy. Evan wanted to keep building web apps while using similar syntax as Angular. He then started the development of Vue.js. He decided to publish his work and made it to the front page of Hacker News. Vue.js became popular due to its ease of use. It's a framework that is easy to learn but has enough to challenge to master. [HistoryOfVue]

Vue is similar to other frameworks, such as React in terms of the utilization of a virtual DOM and reactive components. Vue does perform better when it comes to rendering. It will render sub-components automatically since components are automatically tracked. In React, developers have to add additional keywords to avoid renders of the whole DOM. Vue achieves will only render the necessary components when changed.

Recently, Vue.js became the most popular front-end framework on Github in the terms of stars. One of the reasons Vue.js has picked up in speed in the last few years is due to its easy learning curve. It offers the ease of vanilla javascript syntax that most front-end developers are acclimated to. Other modern frameworks such as React and Angular require the knowledge of ES6 javascript, which has not yet been widely adopted.

Vue.js is also grabbing developer's attention through its flexibility of being adapted into current web applications. Vue.js has the range to be used to build new applications or add onto existing web pages. Since its so flexible, developers can feel comfortable to use it at any level.

Vue.js is also light-weight framework compared to its counter-parts. A majority of users will leave a website if it takes more than three seconds to load the web application. Vue.js ensures a smooth experience for the user with a fast bootup time. [Popularity]

It is possible to use Vue.js without installing it by using their CDN in a <script> tag:

`https://cdn.jsdelivr.net/npm/vue/dist/vue.js`

However, for larger applications, I recommend using npm to download the framework. To use npm, download Node.js first. After installing npm, execute the following in a terminal:

Listing 54: Installing Vue.js

```
npm install vue
npm install --global vue-cli
```

The CLI will be useful for initializing Vue.js projects and running our application server. To get started, type in these commands into the terminal:

Listing 55: Getting Started

```
vue init webpack myVueProject
cd myVueProject
npm install
npm run dev
```

`vue init webpack` creates a webpack module. Webpack will make use of .vue files and makes it easier to use both markup with Vue.js syntax. Vue files also organize the style within a component. Examples will be given once in the hello vue example below.

`cd myVueProject` changes the directory to the vue project that was just created.

`npm install` checks the dependencies in package.json and installs any package that is missing.

`npm run dev` runs the development server and will run the initial boilerplate Vue.js template. Go to localhost on port 8080 to see the initial app. The tutorial will cover some of the boilerplate code after it covers the basics of Vue.js. [Installation]

3.11.2 How to use Vue.js

For a majority of this tutorial, we will review the basic operations and uses for Vue.js without the complete use of a .vue file. Further into the example app is when the .vue file will be used. For the basics, assume this is a .js file.

Vue begins when a Vue instance is initialized.

Listing 56: Initialize the Vue instance

```
var vm = new Vue({
    // additional
})
```

Many developers' use vm since Vue can be associated with the view model design pattern.

The Vue instance takes in a JSON object as an argument. There are many options that can be passed in. The first and most common option is the data object.

Data can be passed into a Vue instance like this:

Listing 57: Data option

```
var data = {fun: false}

var vm = new Vue({
    data: data
})

// vm properties can be accessed like this now
vm.fun == data.fun // when false == false, it returns true
```

Since the view is reactive, changing a data element will re-render the view model. To access instance properties and methods, use the $ symbol like so:

Listing 58: Instance Methods

```
vm.$data === data // => true
```

Take advantage of all of Vue's instance methods using its API reference.

3.11.3 Vue Templates

One of Vue's top highlights is the template syntax. Instead of using JSX like React.js (which requires prior knowledge of ES6), Vue uses templates that mimic HTML syntax. Data can be interpolated using the "double mustache" syntax: `{{ }}`

Listing 59: Templates

```
//vm.title = "Hello World!";
<h1>Title: {{ title}}</h1>

//javascript can be executed inside the mustaches.
//vm.counter = 0
{{ counter + 1 }}
```

A powerful way of using templates is taking advantage of Vue directives. A directive is a special HTML attribute using `v-` such as `v-if`. If data has a list, `v-for` can be used to iterate through it and easily repeat HTML elements.

Listing 60: v-for

```
<ul id="example-1">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

Vue lists

Using `v-model` is a powerful way to use Vue's reactive elements. It enables two-way binding of data. Try the following for example:

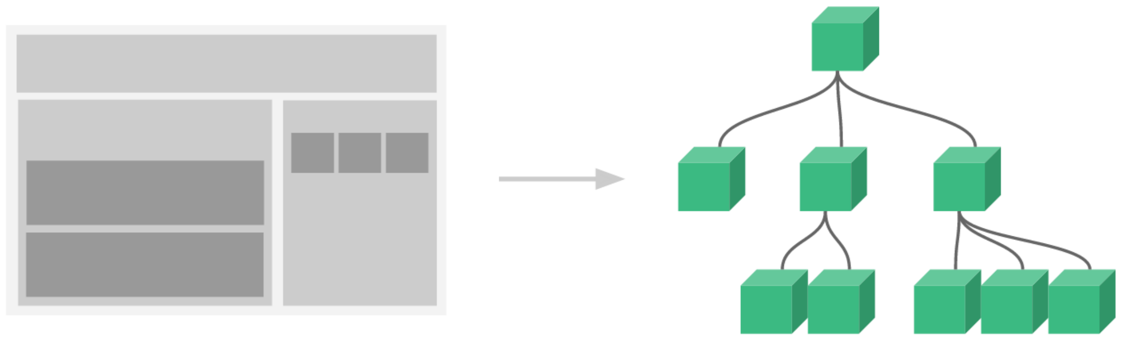
Listing 61: v-model

```
<div id="app-6">
    <p>{{ message }}</p>
    <input v-model="message">
</div>

var app6 = new Vue({
    el: '#app-6',
    data: {
        message: 'Hello Vue!'
    }
})
```

Changing the input field will change the message since it's a two-way binding system.

3.11.4 Components

Using Vue components is the core of its framework. Components are created into reusable HTML elements. Splitting up the application by components is a common practice for front-end development. Components will encapsulate other components starting at the root level. Think of components being a tree structure starting with the root, usually denoted as the <App/> element, then branching off with other child components. Vue.js provides an implementation of components through the Vue instance. [Tutorial]



Components usually consist of a template and optional Vue functions. A Vue component can be created like this:

Listing 62: Component

```
Vue.component('root-app', {
data: function () {
    return {
    message: "Hello World!"
},
template: <h1> {{ message }}</h1>

}})
```

(continues on next page)

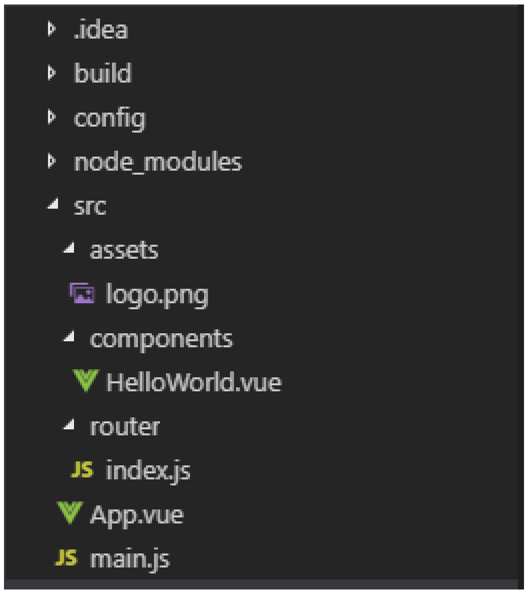
(continued from previous page)

```
//Inside the html
<div id="demo">
    <root-app></root-app>
</div>

// In the js
new Vue({el: '#demo});
```

When initializing Vue instances, use the el option to associate the instance with the div id inside the html. In the vue components, please note the template line. With the template defined, vue will inject `<h1> {{ message }}</h1>` into <root-app> elements within the component.

3.11.5 Basic Hello Vue Example

In this example, the project structure will be covered and some basic syntax will be explained. This involves more advanced syntax. This example is more tailored for developers who want to build larger applications. To get started, refer to the installation at the beginning of this article. First and foremost, here is a picture of what the directory structure should look like:



```
.idea
build
config
node_modules
src
  assets
    logo.png
  components
    HelloWorld.vue
  router
    index.js
  App.vue
  main.js
```

The file that bootstraps the application is main.js:

Listing 63: main.js

```
import Vue from 'vue'
import App from './App'
import router from './router'

Vue.config.productionTip = false

new Vue({
    el: '#app',
    router,
    components: { App },
```

(continues on next page)

(continued from previous page)

```
    template: '<App/>'
})
```

Notice the `<App/>` html element. This is the root of the application. Add to the components according to the amount you have in the current template. Since this is just the root level of the application, there is typically only one component.

Now let's take a look at .vue files, starting with App.vue:

Listing 64: App.vue

```
<template>
    <div id="app">
    <img src="./assets/logo.png">
    <router-view/>
    </div>
</template>

<script>
    export default {
    name: 'App'
    }
</script>

<style scoped>
</style>
```

Typically .vue files have three parts: template, script, and style. This file represents a component. It has a template and the script attached to it. Newer to Vue.js is the `<router-view/>`. This is a special element that takes a peek at the index.js in the router folder of the project:

Listing 65: index.js

```
import Vue from 'vue'
import Router from 'vue-router'
import HelloWorld from '@/components/HelloWorld'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'HelloWorld',
      component: HelloWorld
    }
  ]
})
```

Whatever the current path is, it takes a look at the name and components and loads the appropriate components. The HelloWorld.vue file would be loaded into the root level of the application. [Tutorial]

3.11.6 Conclusion

If you are looking to enter the world of front-end development, then Vue.js is a great way to start learning. As seen in several examples listed in this article, anyone who has basic html and javascript knowledge can start Vue.js today. For advanced users, they can also use Vue.js to build lightweight applications. New users can use the CDN to use Vue.js and advanced users can use Vue-CLI and Webpack to build larger applications. The biggest advantage that Vue.js has over other frameworks is the ability to be adopted at any level of an application, from the ground up or it can be incorporated into current applications.

3.11.7 Citations

3.12 NodeJS

3.12.1 Intro

Node.js is a helpful run-time environment that is able to quickly and easily run JavaScript code and this tutorial will show how to set up Node.js from with things like, installation, REPL commands, Node package manager, and setting up Node.js with MySQL for database queries.

Node.js is a server-side platform built from Google Chrome's JavaScript Engine. Node.js is open source, cross-platform runtime environment for developing on the server-side. Node.js is written in JavaScript and can be ran on OS X, Microsoft, Windows, and Linux. [Tutorialspoint]

Node.js also provides a library with many JavaScript modules which simplifies the creation of web applications.

Node.js uses non-blocking, even-driven input/output to remain efficient in data- intensive real-time applications that runs across many devices.

3.12.2 History

Node.js was first created by Ryan Dahl and was developed and maintained by Dahl which later got sponsored and supported by Joyent. Dahl created Node.js because he had a distaste about the way Apache Http server used to handle a lot of concurrent connections and how the code being created was either blocked by the entire process or implied multiple execution stacks. [Thinkmobiles]

3.12.3 Getting started on Installing Node.js

Starting off, if you're installing Node.js on Windows then you can just go to nodejs.org and download, follow the prompts and you're set. A thing to take note is that the default path that it is installed at is `C:\Program Files\nodejs\bin` that will be the Node.js directory.

On the other hand if the user installing on UNIX/Linux/Mac OS X or SunOS they'll first need to download and extract the archive into /temp, and move the extracted files into a specified directory directory for Node.js.

Listing 66: Command Code

```
$ cd /temp
$ wget http://nodejs.org/dist/v10.15.3/node-v10.15.3-linux-x64.tar.xz
$ tar xvfz node-v10.15.3-linux-x64.tar.xz
$ mkdir - /usr/local/nodejs
$ mv node-v10.15.3-linux-x64/* /user/local/nodejs
```

To make sure it's installed and working validate it by executing a file. The user can easily do this by making a file like test.js on their machine and have some test code like following.

Listing 67: Test for Installation

```
console.log("This is a test.")
```

After that execute test.js on the Node.js interpreter to see the result.

Listing 68: Executing test

```
$ node test.js
```

If it is installed correctly it should print.

Listing 69: Executing test

```
This is a test.
```

3.12.4 Getting an Application Started

Next up to create the server you will have to call on the http module and then use that to create a server and bind it to a port.

Listing 70: First Application

```
var http = require("http");

http.createServer(function(request, response){
response.writeHead(500, {'Content-Type': 'text/plain'});
response.end('Test');
}).listen(8080);
console.log("First application instance");
```

Line one uses the require directive to store the returned HTTP instance into an http variable from http module. On line three is where you create an http instance by calling the `http.createServer()` method that creates the server instance and then on line six you bind it to port 8080. By default, once the user starts the server it'll automatically go to `http://127.0.0.1:8080` in a web browser. The result should be what you put into `response.end()` on line five. To stop the server instance, just hit `Ctrl+c` in the command line.

3.12.5 Node.js Virtual Environment

Node.js comes with a virtual environment called REPL (also Node shell). REPL is the abbreviation of Read-Eval-Print-Loop. Its a way to quickly test simple Node.js/JavaScript code.

To start up REPL is as easy as just typing node into the command console. After typing node and hitting enter, the user will able to run JavaScript, use variables and multiline expressions.

REPL Commands

Command	Action
ctrl+c	Terminates current command
ctrl+c twice	Terminate the Node REPL
ctrl+d	Terminate Node REPL
up & down keys	See command history and modify previous commands
tab Keys	List of current commands
.help	List all commands
.break	Exit from multiline expression
.clear	Exit multiline expression
.save “filename”	Save current Node REPL session
.load “filename”	Load file content into current Node REPL session

Note: As an add on to get the last result, “_” can be used to get that.

3.12.6 Node Package Manager (NPM)

NPM has two main functionalities: online repositories for node.js packages and modules, and command line utility to install Node.js packages, and do version management and dependency management.

To check the current version of NPM just do the type the following in the command console-

Listing 71: Checking Version

```
npm --version
```

If it is an old version of NPM the user can update using the following command-

Listing 72: Update NPM Version

```
$ sudo npm install npm -g
/usr/bin/npm -> /usr/lib/node_modules/npm/bin/npm-cli.js
npm@6.4.1 /usr/lib/node_modules/npm
```

Code used from [Tutorialspoint]

Installing Modules

Next up to install a module and use it in a JavaScript file, in the command line type-

Listing 73: Install prompt

```
npm install <Module Name>
```

Next go to JavaScript and type in the following-

Listing 74: Using the Module

```
var variableName = require('/path/to/file')
```

In the code above we use the require function which is a module in Node that is on a global scope so it's always available. The require is the command that calls on the modules where they're located locally. [Buna]

By default NPMs installs any dependency in the local mode. Where the local mode refers to the package installation in node_modles directory in the folder where Node is present. To globally install a module use-

```
npm install "modulename" -g
```

This will store the packages and dependencies in system directory and cannot be imported using require() in the Node application directly.

Note: To check all modules installed us `npm ls` and `npm ls -g` to check globally installed packages.

Uninstalling, Updating, Searching a Module

Uninstalling, updating and searching a module is simple and can easily be done by doing the following -

Listing 75: Update, uninstall, search

```
$ npm uninstall "ModuleName"

$ npm update "ModuleName"

$ npm search "ModuleName"
```

Package.json

package.json is in the root directory of any Node application/module and is used to define properties of a package.

Attributes of Package.json

- Name - name of the package
- Version - version of the package
- Description - Description of the package
- Homepage - Homepage of the package
- Author - Author of the package
- Contributors - Name of contributors to the package
- Dependencies - List of dependencies
- Repository - Repository type and URL of the package
- Main - Entry point of the package
- Keywords - Keywords

Creating Modules

Now that the basics are done, the user can also create a module. This requires package.json to be generated. Using NPM will generate a basic skeleton of package.json.

Listing 76: Create modules

```
$ npm init

npm help json
npm install <pkg> --save
^C
Name:(webmaster)
```

After `$ npm init` is used, the command prompt will walk the user through making a package.json file that covers common items. For help for package.json documentation, `$ npm help json` will show definitive documentation on the package.json fields and what they do. `$ npm install <pkg> --save` will install the package and save it as a dependency in package.json file.

Next up is registering the user with the NPM repository site using a valid email address. This can be done by doing the following-

Listing 77: Publishing modules

```
$ npm adduser
Username: "Your username"
Password: "Your password"
Email: "Your email"
$ npm publish
```

`npm publish` is when the user actually publishes the modules, but before that a valid account is needed. An important thing to note is that the email address will be public and on the internet and in the fields where it says "Your . . ." put in the respective username, password, and email for the user that is publishing the module.

3.12.7 Setting up Node.js with MySQL

There are many ways to set up Node.js with a database and it may seem complicated, but it's actually simple and this quick tutorial will show how to connect Node.js with MySQL.

First up, how to install the MySQL module. This can be done in the command console. After the module is installed, the next step is to make a JavaScript file that creates the connection and what will be used to query the database.

Listing 78: Installing MySQL module

```
npm install mysql
```

Listing 79: Connecting to a database

```
var mysql = require('mysql')

var con = mysql.createConnection({
    host: "localhost"
    user: "yourUserName"
    password: "yourPassword"
    });

```

(continues on next page)

(continued from previous page)

```
    con.connect(function(err){
    if (err) throw err;
    console.log("Connected")

    con.query(sql, function (err, result){
    if(err) throw err;
    console.log("Result: " + result)
    })
})
```

[W3Schools] used as reference code

Where the user made the variable `con` is where the user will create a connection, and this means you'll have to enter the correct information about the database, such as the host, user, and password. After that the function after will make the connection and handle any errors. With con.query() how the user makes the statement they want is to replace is by replacing the part of the code where it says `sql` in the con.query() function. The whole sql statement will have to go before the function is called.

3.12.8 Conclusion

In conclusion, Node.js is a helpful tool for quick testing of a javascript file that can be helpful and reliable. The best thing about Node.js is the amount of modules a user can get to help the person do what they need.

3.12.9 Sources

3.13 AngularJS

AngularJS is a JavaScript front-end framework program that helps with the development process. Building dynamic single page applications that are interactive and versatile and best for professionals. Angular might take people longer to learn than other frameworks, but if you are developing a data-driven, large-scale application with complex logic, it allows you to work on the logic and get a great running page in the browser. Overall AngularJS is a program that allows you to design large scale frameworks all while minimizing the code that the coder must write due to how the program works.

3.13.1 History

AngularJS was started in 2019 by a Google employee named Misko Hevery. Hevery wanted the program to help with front-end and back-end application. The idea turned out very well, and the project is now officially supported by Google. AngularJS version 1.0 was released in 2012. [Huszárik]

When learning AngularJS you should already know HTML, CSS, and JavaScript.

Listing 80: AngularJS is written in JavaScript.

```html
<!-- AngularJS is used in a JavaScript file and need this tag -->
<script type="text/JavaScript" src="code.angularjs.org/1.7.8/angular.min.js"></script>
```

3.13.2 MVC Framework

Model View Controller is a web application software made up of three parts. The first level is simple script which maintains the data, is called the model. The view is how the data is presented on the screen and how that data changes by the user moving on the screen. The controller interacts with both the model and the view. The controller happens in the view and checks it with the model. It then updates the web application. [Lau] [Ray]

3.13.3 Interface HTML

Using HTML for the user interface simplifies app development and keeps the code easy to understand and well structured. HTML makes it easy to understand the style of the tag. HTML is a declarative language that makes it simple to organize and less likely to break, unlike other app development software that uses JavaScript Interfaces. AngularJS and HTML can find what you want with dependencies making it a lot easier to understand program flow and loading. [Lau] [Ray]

3.13.4 POJO

POJO stands for plain old JavaScript and this goes back to the model part of MVC in which AngularJS binds pieces together without using a getter or setter. Objects can have loops created with properties, which developers can then make changes directly to the loop and adjust the re-frame, making the code cleaner and more precise. POJO is different than traditional data models because Angular's data model has a middle holding area that works with the controller and the view to collect data from the user. The middle area stores data and looks for changes to the stored data, along with updating the view spontaneously. [Lau] [Ray]

3.13.5 Two-way binding

Two-way binding makes the application process simpler, along with difficult manipulations and calculations. The view and the model work together with data to make them sync automatically. With AngularJS you can bind different elements and in the correlation, the view and the model will still work together and keep the page updated correctly. [Ray]

3.13.6 Filters

Filters can be used to do formatting of numbers and arrays based on specific parameters that can standalone. A standalone function is useful to have when creating an app to keep your code clean and organized. AngularJS will also let create your own filter just like the directive `app.filter('newFilter', function()` filer examples [w3schools] [Lau]

- currency
- data
- filter
- json

- limitTo
- lowercase
- number
- orderBy
- uppercase

```
<p>The name is {{ lastName | uppercase }}</p>
```

The filter filer will return anything in the array containing the item that was being searched through. For example if you had a list of name and searched the letter ‘b’ it would return everything with the letter ‘b’. [w3schools]

3.13.7 Routing

Routing is important to the single page applications (SPA) and updating from one view to the next. Developers don’t want to load a whole new page when something is clicked so they us the same page and just change the URL. With AngularJS, changing views becomes easier with single page applications. Routing is what happens when the user changes the view on the screen and the new page should load while changing the URL. This method should make the user think they are interacting with the site. For example, when you are on a website and you select the menu button it should be like you are interacting with the page, rather than loading a whole new page when you just want to access the menu button on part of the page. With AngularJS you can create multiple views for URLs allowing the website transition from one view to another seamlessly. [Ray]

Listing 81: ngRoute

```
<!-- The $routeProvider will have the different routes to your page -->
<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
    $routeProvider
    .when("/", {
        templateUrl : "html1.htm"
    })
    .when("/red", {
        templateUrl : "html2.htm"
    })
});
</script>
```

3.13.8 Directives

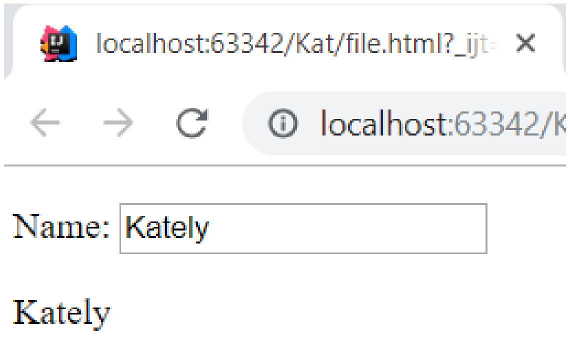
Directives is creating a unique tag specific to your code and should be a standalone element that is separate from the app. This tag can be used anytime once it is defined and will work similarly, just like any other tag in HTML. Creating a new element is possible by putting the DOM manipulation code into directives and give them custom attributes and class names. MVC app can now look directly and the new data and update it with the view. [Lau]

Extending HTML

In this example of code AngularJS works with the HTML using `ng-directives`. The code applies AngularJS with adding `ng-app=""` to the div tag. Next in the input field `ng-model` binds the application data which is name to the `ng-bind` binds is the view we see on the screen. [w3schools]

Listing 82: ng-directives with HTML

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></
↪script>
<body>

<div ng-app="">
    <p>Name: <input type="text" ng-model="name"></p>
    <p ng-bind="name"></p>
</div>

</body>
</html>
```



Name: Kately

Kately

Creating Directives

A directive can be created for elements, classes, and attributes. This simple example will be for an element name that will print out a `<h1></h1>` tag, but with the functions you are able to do more complex things. One thing to be careful with is the name of the function and the tag that must follow correctly for to work. The tag should use - to separate the words and the function name should the same just camel case. [w3schools]

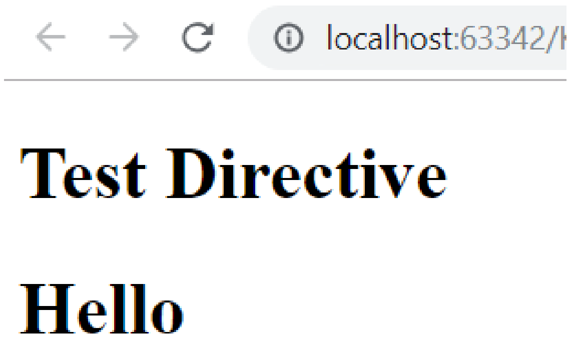
Listing 83: creating a directive

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></
↪script>
<body ng-app="myApp">

<h1>Test Directive</h1>
<script>
    var app = angular.module("myApp", []);
    app.directive("testDirective", function() {
        return {
            template : "<h1>Hello</h1>"
        };
    });
</script>

```

(continues on next page)

(continued from previous page)

```
<test-directive></test-directive>

</body>
</html>
```



Test Directive

Hello

3.13.9 DOM

Document Object Model (DOM) adds behaviors, and with Angular it should be inside directives. Therefore, the user interface designers can see the view without all of the behaviors. [Lau]

Listing 84: HTML DOM elements

```
<!-- This is a simple example but could be added to a button-->
<div ng-app="">
<p ng-hide="true">I am not visible.</p>
<p ng-hide="false">I am visible.</p>
</div>
```

3.13.10 Conclusion

Today people want websites and app to work fast and be interactive and AngularJS achieves that. AngularJS framework was developed to improve the development process and use less code. AngularJS is a good framework because of flexibility and how everything connects and works with the MVC.

3.13.11 Citations

3.14 Three.js

Three.js is a 3D graphics API (application programming interface) for JavaScript [Threejs]. It is meant as a tool for higher-level programming in comparison to pure WebGL. It has gained popularity in a variety of projects, including online games, demonstrations, and models. Three.js gives programmers an opportunity to develop 3D programs that are organized and readable, while also offering a significant amount of features to create fantastic projects. This document will provide the benefits of the Three.js library, along with a basic example and explanation of how to use Three.js.

3.14.1 History

Ricardo Cabello's first release of Three.js on Github was on April 23, 2010 [Github]. Two other contributors, Branislav Ulicny and Joshua Koo, were quick to contribute with their own progress to the project by contributing to geometry, materials, and post-processing. In the last nine years since Cabello's first release, there have been over 25,000 commits with assistance from over 1,000 individual contributors.

3.14.2 What Three.js Adds

Three.js is significantly easier to learn than pure WebGL [Mozilla]. The lower barrier to entry is incredibly helpful for users who are new to developing 3D projects. For experienced developers, Three.js makes it easier to create the same projects in less time and with less effort. Three.js is specifically meant for developers who want to work with graphics and animation without having to worry about how it will interact with hardware. A new developer does not have to learn any WebGL if they want to start developing a 3D application [Threejs]. Instead, they can create the same projects with Three.js in less code and time.

3.14.3 Three.js Example

The following is a basic example of what can be done using Three.js. There are three sliders that allow the user to change how the cube rotates on the three different axes.

3.14.4 Explaining the Code

There are a lot of pieces of this code, but when broken down, it is much easier to understand what is going on to create this project. To start, we create the camera, the scene, and the renderer.

```
//We use this to set how big of a screen we want our camera to see.
var innerDiv = document.getElementById("threejscanvas");

//Creates a new camera.
var camera = new THREE.PerspectiveCamera(70, innerDiv.offsetWidth / innerDiv.
↪offsetHeight, 1, 1000);

//Sets the "z" coordinate of the camera to 500. By default, the x, y, and z␣
↪coordinates are 0.
camera.position.z = 500;
```

The camera is the same as a camera in real life; it is used to view the project we are creating. Depending on the angle and position we place the camera, we can see different parts of our project. In this example, we will not be modifying the camera any more after this.

When initializing the camera, we set field of view, aspect ratio, near plane, and far plane values. In this example, we don't utilize any of these besides the aspect ratio, which we ensure is the ratio our canvas is. The default location of the camera is (0, 0, 0), so we change the z value to 500. This will allow us to see our cube that we will be positioning at (0, 0, 0).

```
//Creates a new scene for adding objects
var scene = new THREE.Scene();
```

The scene is where we place all of our objects. In our scene, for example, we will be adding one cube with a specific size and position. If our camera is pointing at the objects in our scene, then they will be displayed to the user. We will eventually be modifying the object in our scene by rotating the cube, which will be updated within our scene.

```
//Creates a new renderer for creating the visuals
var renderer = new THREE.WebGLRenderer();
renderer.setPixelRatio(window.devicePixelRatio);
renderer.setSize(innerDiv.offsetWidth, innerDiv.offsetHeight);
```

Finally, the renderer is used to process our scene. This is the part that uses WebGL to actually display the scene within our project. Without the renderer, the scene and project are data that cannot be visualized. We also need to make sure to set the pixel ratio and size, in order to render the objects in a way that they aren't distorted in browsers of different size.

```
//Creates the cube shape
var geometry = new THREE.CubeGeometry(200, 200, 200);

//Creates the material, or texture, for the shape
var material = new THREE.MeshNormalMaterial();

//Puts the shape and material together
var mesh = new THREE.Mesh(geometry, material);

//Adds the newly created cube with a material into the scene to be displayed
scene.add(mesh);
```

Creating an object requires three parts: The geometry, the material, and the mesh. The geometry is the shape of the object we want. It also stores the location of the object. In this case, the object's location is the default (0, 0, 0). In this example, we've created a 200x200x200 cube for our geometry. The material is the texture of the object. For this example, we are using MeshNormalMaterial, which is a basic material that shows a simple color spectrum across the object. However, we could have also added a custom texture in a similar way. Finally, we put them together in the mesh. The mesh is what we add to the scene to be displayed.

```
//Adds the scene to our HTML page
document.getElementById("threejscanvas").appendChild(renderer.domElement);

//Begins animating the scene
animate();
```

These next two lines are incredibly important. The first line adds the scene we have created into our HTML file using the renderer we created before. This is what lets us see the project within our HTML page.

The animate function is used to constantly update our scene. This is where we begin to add animations; in our case, these animations are rotations of the cube. However, we could change anything we wanted within our animate function, whether it be the shape, the material, or the location.

```
//If the window ever gets resized, the size and aspect of the scene will change
window.addEventListener('resize', onWindowResize);

function onWindowResize()
{
        camera.aspect = innerDiv.offsetWidth / innerDiv.offsetHeight;
        camera.updateProjectionMatrix();
        renderer.setSize(innerDiv.offsetWidth, innerDiv.offsetHeight);
}
```

This function is not necessary in all cases, but is important in order to have a responsive webpage. If the window is resized, this function will change the size and aspect ratio. The updateProjectionMatrix function is necessary after changing the aspect ratio of the camera so that it can refresh correctly.

```
function animate() {
        //Used to call animate again
        requestAnimationFrame(animate);

        //Renders the scene based on the view of the camera
        renderer.render(scene, camera);

        //Gets the rotaion in all three axes directions based on the HTML sliders
        xRotation = document.getElementById("xRotation").value;
        yRotation = document.getElementById("yRotation").value;
        zRotation = document.getElementById("zRotation").value;

        //Adds the rotation value to the cube to allow it to rotate.
        mesh.rotation.x += parseFloat(xRotation);
        mesh.rotation.y += parseFloat(yRotation);
        mesh.rotation.z += parseFloat(zRotation);
}
```

Finally, we add the animations that allow the cube to rotate. This is done by calling the animate function each frame, then modifying the values of the mesh we created. In this example, we first take the values from the HTML sliders and put them into three variables: xRotation, yRotation, and zRotation. Next, we add them to the corresponding rotation values of the mesh. This gets updated every time animate is called by rendering the scene again, which we also do within this function. Anything pertaining to modifying the scene can be done within this function.

3.14.5 Benefits of Three.js

Besides the ability for users to develop WebGL programs in an easier, higher-level language, there are many other benefits to using Three.js. First, Three.js provides fantastic examples of how to use their library, making the barrier for entry even easier [Threejs]. With hundreds of examples available, it is very likely a developer can find the feature they are looking to add. In addition, Three.js continues to be constantly updated, meaning there won't be any issues of the library being outdated in the near future [Github]. Finally, the main Three.js package is very light; developers have the option to add other libraries they may need later on.

3.14.6 Problems with Three.js

While this library excels in a variety of ways, there are still some downsides to using Three.js. There is no official versioning system at this time, meaning the API still ocasionally changes [Github]. This could mean an update to a developer's Three.js library has the potential to completely break any current projects. While it is unlikely this would occur, it is still something that must be acknowledged. Three.js also has an online editor [Threejs]; however, it is poorly implemented and considered by many to be useless. Developers are better off using the examples on the website or generating something themselves. Finally, the documentation for this library is not written very well. It lacks a lot of description that would be helpful to developers who have just started with developing 3D projects. Fortunately, the examples provided on the website do a decent job of making up for the lack of documentation. While there is a lot that Three.js offers in their API, they could ultimately do some more work on how they showcase this work to the developers using it.

3.14.7 Conclusion

Three.js is a great library for users who are beginning their journey into 3D development, as well as experienced users who are looking for a powerful library. The higher-level programming makes it significantly easier to create projects in comparison to pure WebGL [Mozilla]. Despite a few fallbacks, it is a great library that I would highly recommend to anyone looking to begin developing in three dimensions.

3.14.8 Citations

3.15 ReactJS Scott

React js is a javascript library used for building user interfaces. Within this informational document, I will go over he short history of React as well as its importance, how to use it and its basic funtions and lastly its future opportunity and potential in the world of technology. [React_OS] [React_hackernoon]

3.15.1 History

Early signs of react can be traced back to 2010 in the form of XHP which was a type of PHP released by Facebook with the intention of combating the problematic occurrences of Cross Site Scripting (XSS) attacks. Later down the line a facebook engineer, Jordan Walke, attempted to fix the issue of web applications making too many round-trips to the server by taking the XHP into the browser using Javascript. This resulted in ReactJS. [TheNewStack]

3.15.2 Why ReactJS is important

In usual cases, rendering javascript App Data to a browser is very costly. With React JS being a library that enables web applications that requires very little code to implement, this user interface creator and editor is a very convenient and time saving library to use, not to mention its relatively new presence in the world of technology. [RisingStack]

3.15.3 How to use/ Tutorial

An important thing to note about React is that it updates and re-renders automatically without having to reload the page and uses an XML like syntax called JSX. React and ReactDom are essential in using ReactJS. ReactDom is an essential element in creating React code and running it. It basically serves as the bridge between the document object model and React. [React_hackernoon]

Laying Down The Foundation

First and foremost, when coding in ReactJS you want to import the React and ReactDOM libraries along with the initial HTML setup.

```
    <html>
    <head>
    <script src="https://unpkg.com/react@15/dist/react.min.js"> </script><script src=
↪"https://unpkg.com/react-dom@15/dist/react-dom.min.js">
    </script>
    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
    </head>

As you can see we import the React and React DOM Library through an HTML-like syntax.
```

The next import is named babel, what is that? Babel is actually just as important as the other two imported libraries. What Babel does is compile JavaScript code into a compatible version of the web browser it is being ran on. In doing this, Babel correctly transforms the syntax and adjusts your code to the right format.

```
<body>
    <div id="root"></div>
    <script type="text/babel">
```

An important step in any web application is the implementation of the <div> tag, we set this id to the “root” which is a DOM node and will be managed by our React DOM which was previously imported. This all basically serves as the “on” button to our React application. Here is where we actually implement Babel after importing it into the program, ensuring that it will correctly adjust the JavaScript to the correct format.

Importance Of Components

React runs off of the uses of components, it serves as the Javascript equivalent to a class. In order to initialize a component you have to extend from the “React-Component” class

```
class Welcoming extends React.Component {
    render() {
        return <h1>Welcome to DC's introduction to React!</h1>;
    }
}

ReactDOM.render(
    <Hello />,
    document.getElementById("root")
);
```

Here we create our class named Welcoming which returns a statement by extending it through another class that is called the React.Component. All we do within that is specify what our class we are extending it from will do, simple stuff. This brings us to the relevance of what render() does, render is a method for the React component that basically renders on the page whatever the user puts within the ReactDOM.render() application. All document.getElementById “root” entails is the linking from our class extending React.component to the div tag.

Data Types

```
ReactDOM.render(
    <Welcoming status="at Simpson College in the Spring" />,
    document.getElementById("root")
);
```

One of the most notable aspects of React are its use of Props. Props are one of the two types of data in React. A prop is acts mostly as any class that can be reused. The code above is how you initialize a prop, in this case my prop details the when and where of the person giving the welcoming statement.

```
class Welcoming extends React.Component {
    render() {
        return <h1>Welcoming {this.props.status}!</h1>;
    }
}
```

As you can see we can call our prop via this.props then the prop we initialized earlier. What is rendered on the page is a concatenation of our initial Welcoming message as well as the text returned by our prop, in doing this you can see that our prop is interchangeable.

```
class Welcoming extends React.Component {

    constructor(){
        super();
        this.state = {
            status: "at Simpson College in the Spring (from state)!"
```

(continues on next page)

(continued from previous page)

```
        };
    }

    render() {
        return <h1>Hello {this.state.message}!</h1>;
    }
}
```

One downside to a Prop is that it cannot be changed directly by a component. This is doable when using a State which is the second of the two data types used in React. In the code above we use the constructor method to setup this.state with our preset "key" which is what we already have as status. Doing what we have so far, the state is initialized.

This is all there is to initializing and using the bare basics of ReactJS. [React_code_tutScott]

3.15.4 Notable Uses/Examples

ReactJS is a JS library used within various popular social networking applications such as Instagram and of course Facebook. A similarity these applications share are its ability to consistently stay up to date by the use of reusable data in combination with ever changing user input. There is a lot of opportunity for future potential with ReactJS especially given its support with the many apps that use it very noticeably the convenient and easily accessible Uber. ReactJS is a simple tool that will likely stay relevant within the world of technology and within the topic of technological growth. [React_FutureScott]

3.15.5 Conclusion

React is a helpful JavaScript Library that stresses convenience and ease of access. That is, in my opinion, what makes React so prevelant and why I believe that it is on the rise. It makes writing complex code for applications more simple and less time consuming. From learning about this Library and what it has to offer I can confidently say that this will definently continue stay relevant in the future.

3.15.6 Sources

3.16 AngularJS

JavaScript is a programming language that is designed to enhance web applications. Since it's release, programmers have been learning and contribute to the pool of frameworks that make writing code easier. AngularJS is one of those frameworks that has proven to be a very powerful. AngularJS that has the ability to two way bind data, create beautiful single page applications, and provides tons of functions for the programmer.

3.16.1 History

AngularJS was initially created by two Google employees: Misko Hevery and Adam Abrons. The two employees originally called the framework 'GetAngular', save web programmers more interaction between the front and back end of the application.

Eventually Hevery had to work on a different project for Google called Feedback. over the course of 6 months there was 17,000 lines of code written for this program, making it increasingly difficult to test. This drove Hevery to rewrite the software but this time using his part of the GetAngular project from earlier. With GetAngular he was able to turn the initial 17,000 lines of code into just 1,500 in 3 weeks. With this massive rework and elimination of lines of code his manager took interest in GetAngular which would lead Google to start heavily development into Angular.js. [Austin]

Over the years, AngularJS would turn into a framework monolith in the JavaScript community. The ability to create single page applications in just a couple hundred lines of code was incredible.

MVC (Model, View, Controller)

A Model-View-Controller architecture is a design pattern for software engineers to help separate the functionality of a application. The application can be separated into 3 main sections.

3.16.2 Model

This section handles how the user data is handled. The data could come from the user or a database. The Model could be considered the JavaScript or any language that will do logical manipulation.

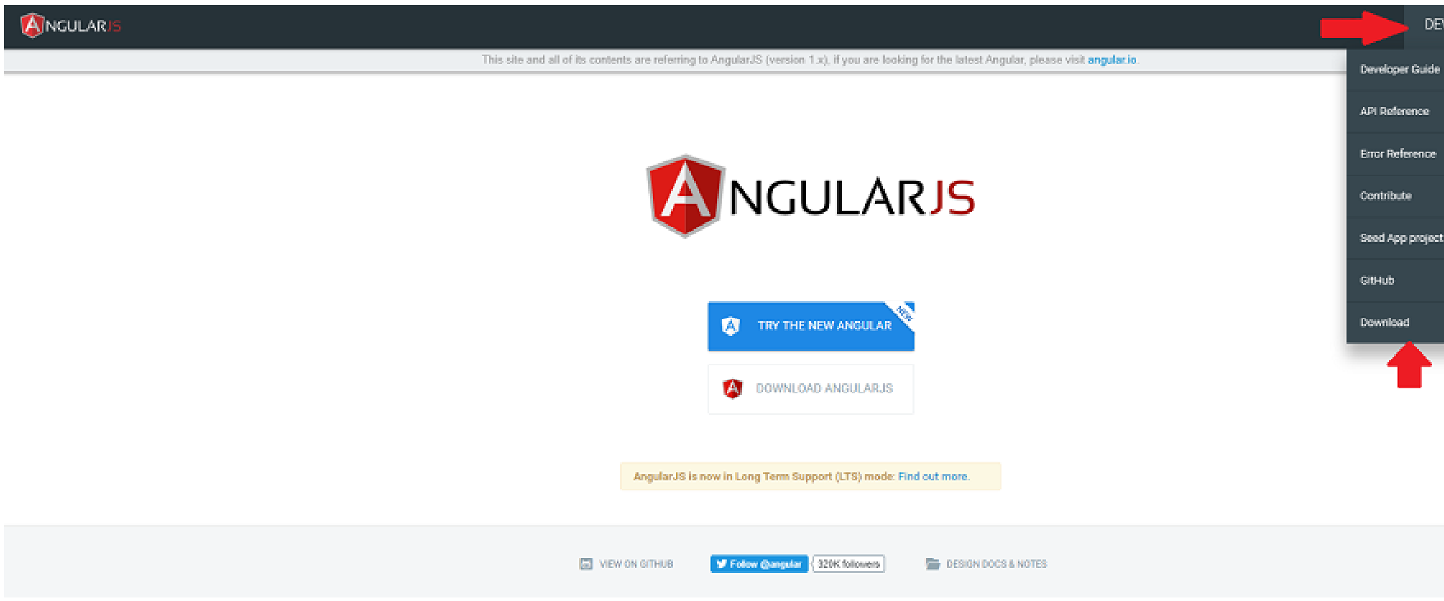
3.16.3 View

The view section is responsible for handling anything the user will visually see. The user interface logic could get data from the controller and send data to the model. The View is usually the HTML.

3.16.4 Controllers

The controllers of the MVC architecture act as a interface for the view and model sections. The controller will manipulate data that has came from a model or some other source and send it back to the model and view . [TutorialspointAngularMVC] [TutorialspointAngular]

3.16.5 AngularJS MVC


This site and all of its contents are referring to AngularJS (version 1.x), if you are looking for the latest Angular, please visit angular.io.

AngularJS is now in Long Term Support (LTS) mode: Find out more.

Starting development in AngularJS is simple and easy. Go to Angularjs.org and finding the version of AngularJS you want. When you find the version you like then grab the URL of the file and stick it in your HTML file. [Angular]

Listing 85: Adding the script for AngularJS

```
 <!-- The src= is where the URL goes-->
<script type="text/JavaScript" src="code.angularjs.org/1.7.8/angular.min.js"></script>
```

Then you need to be worried about the global namespace that Angular uses. The framework has tons of preset global variables which could interfere with your JavaScript.

3.16.6 Global Namespace

While creating a HTML document you can incorporate many JavaScript libraries to enhance the document. One fear is that the JavaScript can override each other if they share similar named variables in their global namespace. consider the following examples:

Listing 86: Global Namespace Example 1

```
var person = 'Adam';
var class = 'Advanced Web Development';

function getInfo(){
    return person + ' ' + class;
}
```

Listing 87: Global Namespace Example 2

```
//This will print log 'Mike' even though in the other
//file 'Adam' was in the person variable

var person = 'Mike';

getInfo();
```

The function in the beginning declares person as ‘Adam’ but prints ‘Mike’ when the function is called in the 2nd file. This is because of the global namespace. This is very important to know and understand before delving to far into AngularJS. As mentioned before, AngularJS comes with a ton of pre-defined variables in the global namespace which can get messy, causing errors and bugs. To combat this the user will have to create their own namespace. One way to create your own namespace is by treating globals you may want to use as a JSON variable.

Listing 88: JSON namespace

```
var myNamespace = {};

myNamespace.person = 'Mike';

getInfo();
```

This Example will no longer use the global namespace in the first example and the function should now return ‘Adam’ as intended. This concept will be very important for dealing with AngularJS. [Alicea]

3.16.7 AngularJS Features

AngularJS offer’s so many features that it makes development easy. In Angular the MVC is easy to understand with just a little example.

Listing 89: Making your HTML document a AngularJS Model

```html
<!--This is the View-->
<html lang="en-us" ng-app="myApp">
```

Adding the ‘ng-app’ is Angular’s way of specifying a model/module the programmer will be able to control in JavaScript. You can name it what ever you want, it just has to be the same name in the JavaScript.

Listing 90: JavaScript of declaring a AngularJS Module

```javascript
// This is Model
// The [] in the parameters is a array of dependencies for Angular to work
// with. You can get other services from the AngularJS.org website and
// include them in this array ex 'ngHttp' will allow you to use the
// $http service in the module
// The first parameter is the name you used in the HTML attribute ng-app
var myApp = angular.module('myApp', []);
```

This code will create a variable which is linked to the DOM (Document Object Model). This variable will be how you control the specified HTML document. Now you may want to manipulate it somehow. This is where the controller part of the MVC comes in.

Listing 91: JavaScript of declaring a Controller

```javascript
// This is the Controller
myApp.controller('mainController', function(){});
```

And before this controller will work with anything you must hook it up somewhere in the HTML.

Listing 92: HTML for connecting a Controller

```html
<!--This is where the controller in the myApp.js is connected to --->
<div ng-controller="mainController">
```

Now you have a controller which you can write code in the function block in the parameters and manipulate the DOM. This is a simplified version of the AngularJS MVC model. Notice in the two code examples above that the ng-controller attribute value matches the string in the 1st parameter in the JavaScript code. [Alicea]

3.16.8 Data-Binding

AngularJS is a fantastic framework for binding data in the JavaScript and the HTML DOM. The user of a website can change something in the HTML via a textbox or some field and it will directly change the JavaScript too. AngularJS makes this easy! consider the following:

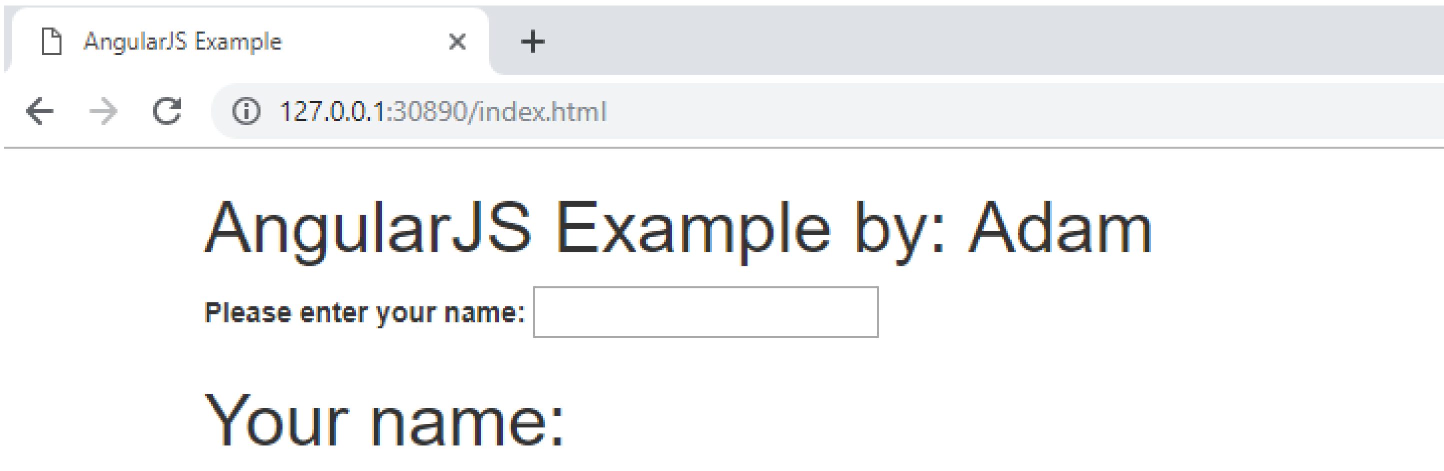
Listing 93: Sample HTML for data-binding

```html
<!DOCTYPE html>
<html lang="en-us" ng-app="myApp">
    <head>
        <title>AngularJS Example</title>
        <meta charset="UTF-8">
    </head>

    <body>
        <div class="container">
```

(continues on next page)

(continued from previous page)

```html
            <div ng-controller="mainController">
                <!-- Angular looks for {{}} and replaces it with anything
                you want to put there. currently there is a
                string called name in the middle of the curly braces
                which will have to match name of the variable in the
                JavaScript you wish to fill it with-->
                <div>
                    <label>Please enter your name:</label>
                    <input type="text" ng-model="name" />
                    <h1>Your name: {{name}}</h1>
                </div>
            </div>
        </div>
    </body>

<script type="text/JavaScript" src="https://code.angularjs.org/1.7.0-rc.0/angular.min.
↪js"></script>
</html>
```



AngularJS Example by: Adam

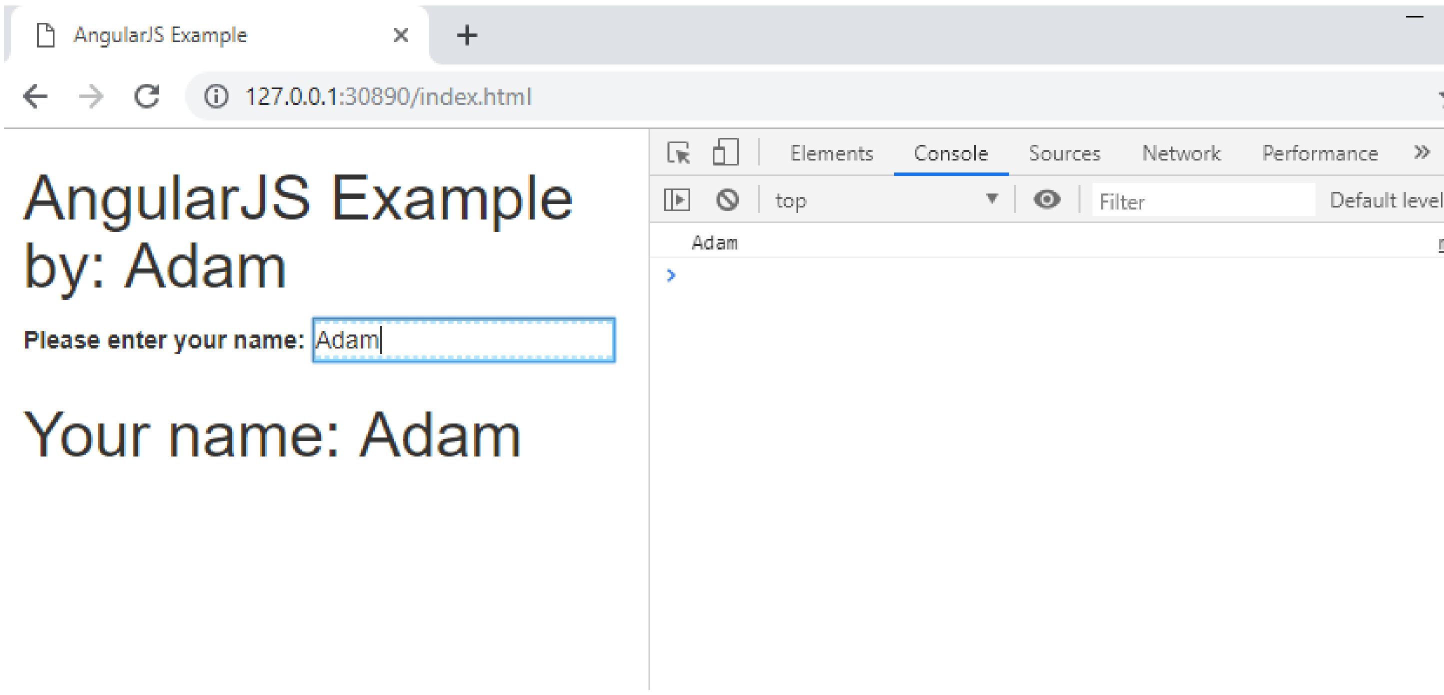
Please enter your name:

Your name:

There is a lot going on in this sample code. in the container div there is a 'ng-controller' attribute which we will link to the JavaScript so we can start manipulating the DOM. This code will connect the two together:

Listing 94: Connecting to the DOM with AngularJS

```
myApp.controller('mainController', ['$scope','$timeout',function($scope,$timeout)
    $scope.name='';
    //$timeout is AngularJS service that can wait x amount of milliseconds
    //before performing a function, in this case I wanted to demo how
    //the two way data binding worked
    $timeout(function(){console.log($scope.name},5000);
)]);
```



AngularJS Example by: Adam

Please enter your name: Adam

Your name: Adam

```
Adam
```

A question that may arise is what is '$scope'. It is important to know that AngularJS prefixes all their variables with either $ or $$ so that the programmer can include more frameworks if they wish. This would help mitigate conflicting names. The $scope variable represents a service offered by AngularJS. There are a whole collection of services that are offered and can be found at Angularjs.org for documentation. The $scope is how Angular 'talks' to the DOM.

3.16.9 Single Page Application (SPA)

Creating a multi-page application can be difficult and costly on the client's browser and the server serving the web pages. AngularJS solves this problem by dynamically changing the users view in the same web page. This eliminates the need to get another web page from the server and doesn't bring along the annoying page stuttering when navigating to a different page.

AngularJS bundles all the code and views into one package and will dynamically load code based on what view the user is currently in. This makes the work load lighter on the server which speeds the website up significantly. Angular takes advantage of the client's computer to load the information on the website.

Testing the web application also becomes much easier when there is only one page to test. This will allow testing suites to more easily test the web application when the development team deploys a new build. This also means rolling back changes is also easy, since everything is bundled together. [Rajput]

Lets take a look on how AngularJS can dynamically change what the user will see. We can add this snippet of code to our HTML file we have from above in the body. It will create 3 links that will let the user change a field in the document.

Listing 95: Sample HTML for Routing in AngularJS

```html
<a href="#/!">Default</a>
<a href="#!Test1">Switch Routes!</a>
<a href="#!Test2">Try a 3rd time</a>
<div ng-view></div>
```

There is also another AngularJS script needed to make it all work.

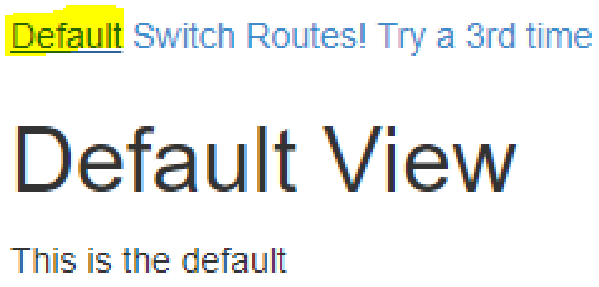
Listing 96: CDN for AngularJS $routeProvider service

```html
<script type="text/JavaScript" src="https://code.angularjs.org/1.7.0-rc.0/angular-
↪route.min.js"></script>
```

Next we need to add a route to our myApp module so AngularJS knows how to navigate through the different potential html pages.

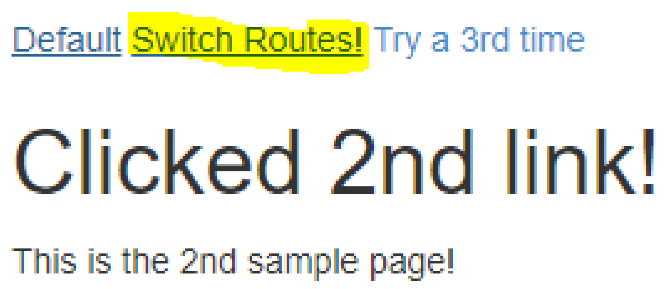
Listing 97: JavaScript to create AngularJS Routes

```javascript
//Add "$ngRoute" into the [] when you create the module
var myApp = angular.module('myApp', ["ngRoute"]);
//This will inject the ngRoute dependency into the module which is not
//included into the default AngularJS library

myApp.config(function($routeProvider){
   $routeProvider
   .when("/", {
       template : "<h1>Default View</h1> <p> This is the default</p>"
   })
    .when("/Test1",{
       template : "<h1>Clicked 2nd link!</h1> <p> This is the 2nd sample page!</p>"
   })
    .when("/Test2", {
       template : "<h1>Clicked 3rd link!</h1> <p> This is the 3rd sample page!</p>"
   });
});
```


Default Switch Routes! Try a 3rd time

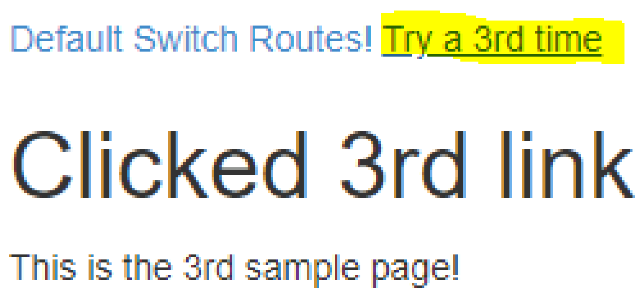
Default View

This is the default


Default Switch Routes! Try a 3rd time

Clicked 2nd link!

This is the 2nd sample page!


Default Switch Routes! Try a 3rd time

Clicked 3rd link!

This is the 3rd sample page!

Earlier when we declared our myApp module the empty array in the parameter list was empty. This is how Angular will inject dependencies into the module (see line 2). These dependencies are usually more services that do not come with AngularJS by default. When working with the routing services we have to get the service delivered through a content delivery network (CDN).Then inject it into our application before we can use it.

Once we have our service we can connect the navigation links in the HTML DOM with our JavaScript to make the magic happen. AngularJS will look at what the URL and then manipulate the DOM based on what it finds. Lines 8 - 14 in listing 13 will look for those specific url extensions and will insert the string of HTML that follows the template into the ng-view attribute in the HTML document. [W3SchoolsAngular]

3.16.10 Chaining Promises = Complicated

AngularJS has some really incredible services and features, one of them being the ability to chain functions together with a concept called called promises. This is important because JavaScript is asynchronous and you aren't really guaranteed an order of operation in some parts of the code. AngularJS can handle many events that may transpire while getting information over some kind of call over the internet. Here is a very simple example:

Listing 98: Exmaple of Promise chaining

```
function returnStudentMajors(){
return $http.get("Some url to get data")
    .success(function(data){
        //do something
    })
    .error(function(data){
        //do something
    })
}
```

This is a simple use of a promise but it can get really complicated when you start chaining them together as follows:

Listing 99: Exmaple of Promise chaining

```
$http.get("Some url to get data").then(function(data){
    //do something
}).then(function(data){
    //do something
}).then(function(data){
    //do something
});
//You can chain this for as long as you have stuff to do on the data
```

It is not important to understand what the $http service is doing in this example, but how the chaining can go on and on because every call returns an object which the programmer can call methods from the returned object. Some chains can get very complex but it can be very powerful. [Strahl]

Disadvantages of AngularJS

AngularJS has a lot of complexity to the framework. One of the annoyances is having to allow JavaScript on your browser to be able to see the AngularJS application. With out JavaScript permissions the page will simply not load. Another major complexity will be the hierarchy of directives that the programmer will have to learn to make optimal use of AngularJS.

AngularJS uses a MVC concept to create single page applications which could be very intimidating and hard if the programmer is not familiar with the concepts before hand. Another confusing aspect is keeping the scopes organized between the views in the web application. Each scope will contain different information depending on what view the user is currently in. [Rajput]

Conclusion

Web development has become one of the hottest areas of tech in today's world. With the increasing demand for web programming companies have spent tons of money on developing frameworks that use JavaScript. AngularJS is one of those frameworks that emerged because a company had some employees that found a purpose from a small side project. Utilizing AngularJS's plethora of services the programmer has everything they need to create a well designed and optimal single page application.

Sources

3.17 ReactJS - MWT

3.17.1 Introduction

ReactJS is a JavaScript library that specializes in building user interfaces. This JavaScript library makes it so a programmer can create user interfaces for their programming application or to update and render the proper components in their application if changes are needed. In ReactJS, you are able to use declarative views which makes your code easier to read and to debug. ReactJS is a component based JavaScript library that focuses on building encapsulated components that are able to manage their own state which can help the user compose some of the most easiest or most complex user interfaces.

3.17.2 History of ReactJS

ReactJS began in 2011 with a couple of software engineers from Facebook. One of those engineers was Jordan Walke who helped integrate ReactJS into Facebook's news feed style set up. After ReactJS was implemented into Facebook's news feed set up, Instagram followed suit as well by adding ReactJS to their application a year later in 2012. After the two social media giants endorsed this JavaScript library, the code was open sourced to the public in 2013. Pete Hunt is another software developer, like Jordan Walke, who has been a core member of the ReactJS software development team.

In an interview with [InfoWorld], Hunt discusses the shift that reactive programming has undergone in the last few years. Hunt stated "We've seen a shift toward what we like to call reactive programming. Meteor and Angular are examples of that. When your data updates, your UI is automatically updated to reflect that, and the system manages that for you. The difference with React is the way that you program it, is much more like a game engine, as opposed to these alternative approaches, with data binding". From this quote, Hunt describes that even though these JavaScript libraries are similar the way they are created syntactically is different, which gives ReactJS a leg up on other competitors.

According to the [TheNewStackArticle], developers from ReactJS also helped minimize malicious Cross Site Scripting (XSS) attacks that occur within the JavaScript language. XSS attacks occur when an attacker enters content that is embedded or hidden within the JavaScript code that is intended to steal or compromise the viewer's information. To prevent these attacks, ReactJS was able to scrub the viewers submitted information, making it so that the attacker comes up empty handed.

When it comes to ReactJS, the developers for the library wanted it to have a different look and presence compared to other popular UI developing languages. ReactJS makes it look like you are coding for a game engine and tries to use syntax that might help the programmer understand what he or she is trying to code.

3.17.3 ReactJS Tic-Tac-Toe Tutorial

This particular block of code derives from the [ReactJSTutorial]. This code lets the user create and interact with a tic-tac-toe board. This particular block of code is organized into different classes. Here is an example of how the first couple of functions operate.

Listing 100: Starting Tic-Tac-Toe: Square & Board Functions

```
function Square(props)
{
    return(
    <button className="square" onClick={props.onClick}>
    {props.value}
    </button>
    );
}
class Board extends React.Component
    {
        renderSquare(i)
        {
            return (
            <Square
            value={this.props.squares[i]}
            onClick={() => this.props.onClick(i)}
            />
             );
        }
    }
}
```

From this code block, a function is created that allows the user to click a button that starts the game. The Board class is created which calls the Square function which allows the user to click a square within the game board. By clicking one of these squares, a user either puts down an X or an O within the square they selected.

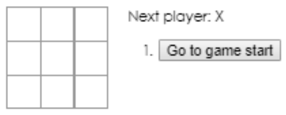
Listing 101: Starting Tic-Tac-Toe: Creating 3x3 Array

```
render()
    {
    return (
        <div>
        <div className="board-row">
            {this.renderSquare(0)}
            {this.renderSquare(1)}
            {this.renderSquare(2)}
        </div>
        <div className="board-row">
            {this.renderSquare(3)}
            {this.renderSquare(4)}
            {this.renderSquare(5)}
        </div>
        <div className="board-row">
            {this.renderSquare(6)}
            {this.renderSquare(7)}
            {this.renderSquare(8)}
        </div>
        </div>
        );
    }

class Game extends React.Component
    {
        constructor(props)
            {
                super(props);
                this.state =
                    {
                         history: [{
                            squares: Array(9).fill(null)
                            }],
                    xIsNext: true
                    };
            }
    }
```

This particular code block creates the array for the tic-tac-toe board. This board is based off of a three by three array with a total of nine different squares. These nine squares are used to simulate one bigger square, thus creating the tic-tac-toe board.

Board Example

This is a visual model of what the two code blocks above produce. This code was able to create a tic-tac-toe board that the user or users can interact with.



Player Input Code

This block of code allows the user to interact with the game board. The handle click function remembers the state of the game board, what box the previous user selected and how a winner is declared. The first user has the ability to select a box with the X marker. Once the user has completed their selection, the second user will have the ability to select their box as well. The second user will select a box with an O marker.

Listing 102: Handling User Input

```
handleClick(i)
{
    const history = this.state.history;
    const current = history[history.length - 1];
    const squares = current.squares.slice();
    if (calculateWinner(squares) || squares[i])
        {
            return;
        }
    squares[i] = this.state.xIsNext ? 'X' : 'O';
    this.setState(
        {
            history: history.concat([
                {
                    squares: squares
                }]),
            xIsNext: !this.state.xIsNext,
        });
}

render()
{
    const history = this.state.history;
    const current = history[history.length - 1];
    const winner = calculateWinner(current.squares);

    const moves = history.map((step, move) =>
    {
        const desc = move ?
        'Go to move #' + move :
        'Go to game start';
      return (
        <li key={move}>
        <button onClick={() => this.jumpTo(move)}>{desc}</button>
        </li>
        );
    });
}
```

After the user has the ability to click on the boxes, a winner and a loser is then implemented into the game. In order for this to happen, refer to the code block below. This particular block of code uses if statements to determine if the user has either won the game or has to keep playing. For the game to find out who the winner is, an array must be put in place that is able to calculate how a user wins. In tic-tac-toe, you can win by getting three X's or O's in a row. To recognize this, this code block detects that once the user gets three in a row, the game will declare a winner and end. If the game ends in a tie, the game will end, letting both users know it was a draw.

Listing 103: Declaring the Winner & Loser

```
let status;
if (winner)
    {
        status = 'Winner: ' + winner;
    }
else
{
    status = 'Next player: ' + (this.state.xIsNext ? 'X' : 'O');
}

return
    (
    <div className="game">
        <div className="game-board">
         <Board
            squares={current.squares}
            onClick={(i) => this.handleClick(i)}
            />
        </div>
        <div className="game-info">
            <div>{status}</div>
            <ol>{moves}</ol>
        </div>
  </div>
);
}

// ========================================

 ReactDOM.render
    (
     <Game />,
    document.getElementById('root')
    );

function calculateWinner(squares)
{
const lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
];
for (let i = 0; i < lines.length; i++)
  {
    const [a, b, c] = lines[i];
    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c])
     {
        return squares[a];
     }
  }
```

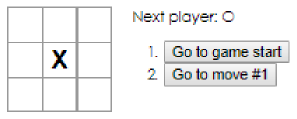
(continues on next page)

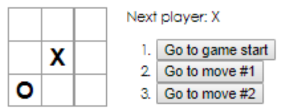
(continued from previous page)

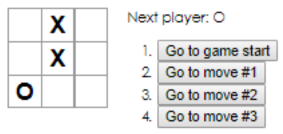
```
return null;
}
```

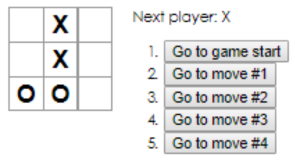
Player Input Example

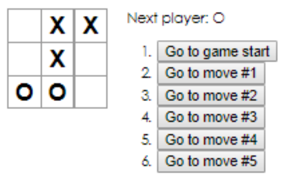
Once all of these code blocks are implemented, the game will work properly. The tic-tac-toe game allows the user to switch back and forth between inputting X's and O's. This will continue until a winner is declared or the game ends in a tie. Here is an example of how a typical game would look like if the user were to win the game.

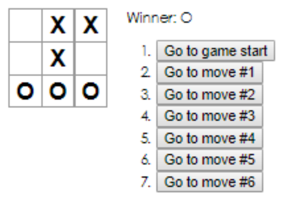












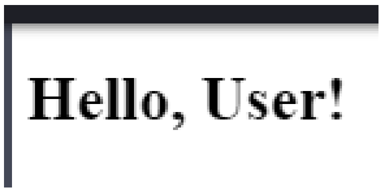
3.17.4 ReactJS Notable Features

When it comes to ReactJS there are some distinguishable features that the JavaScript library has. One of these features would be the [ReactJSComponents]. Components in ReactJS are able to split up the user interface into independent reusable pieces. Components are similar to JavaScript functions. Components are able to accept inputs and return elements that are able to describe to the user what they should be seeing on their screen. Here is an example of what a simple component looks like.

Listing 104: Simple Component Example

```
function Welcome(props)
    {
        return <h1>Hello, {props.name}!</h1>;
    }

const element = <Welcome name="User" />;
ReactDOM.render(element, document.getElementById('root'));
```

This particular code block is able to display a simple message to the user. The function component is a welcome message. An HTML tag is created to display the message and a props element is made to be called in the function. In this case the name for this prop's element is called user. This is the output of the code.



```
Hello, User!
```

Another notable feature of ReactJS would be the [ReactJSHandling]. Handling in elements is similar to handling DOM elements in ReactJS. DOM stands for Document Object Model and is used to show the programming interface. When it comes to handling events in ReactJS, there are syntax differences compared to using JavaScript. In this case a component is being used within a class which is a common method for an event handler to be within a method of a class. In this example, the toggle component is being used within the class. The toggle class utilizes the handle click and constructor methods to tell when the button is on and when it is off.

Listing 105: Simple Handling Example

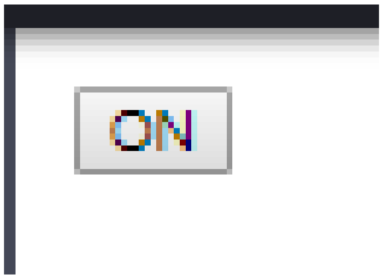
```
class Toggle extends React.Component
    {
        constructor(props)
            {
                super(props);
                this.state = {isToggleOn: true};

            // This binding is necessary to make this work in the callback
                this.handleClick = this.handleClick.bind(this);
            }

        handleClick()
            {
                this.setState(prevState => ({
                isToggleOn: !prevState.isToggleOn
                }));
            }

        render()
            {
                return (
                <button onClick={this.handleClick}>
                {this.state.isToggleOn ? 'ON' : 'OFF'}
                </button>
                );
```

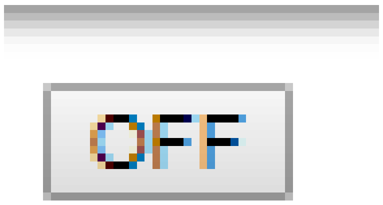
(continues on next page)

(continued from previous page)

```
            }
    }

ReactDOM.render(
<Toggle />,
document.getElementById('root')
);
```

This is the output for what this code block can produce. As you can see the handling used for this button was able to toggle the button to turn it on or off. This is toggle is initiated once the button is clicked on or when it is clicked off.





Another notable feature in ReactJS would be the [ReactJSForms]. The forms in ReactJS are similar to the HTML forms we worked on in class. However, with ReactJS you are able to use JavaScript functions and HTML code to create a technique called controlled components. These components are able to be used with specific HTML tags. In this example, the form is able to use a value tag with a select tag which is able to help update the form more conveniently. The render method implements these tags and compiles it with the other methods in the FlavorForm class.

Listing 106: Simple Form Example

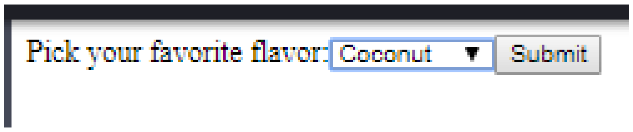
```
class FlavorForm extends React.Component
    {
        constructor(props)
            {
                super(props);
                this.state = {value: 'coconut'};
                this.handleChange = this.handleChange.bind(this);
                this.handleSubmit = this.handleSubmit.bind(this);
            }

        handleChange(event)
            {
                this.setState({value: event.target.value});
            }
```

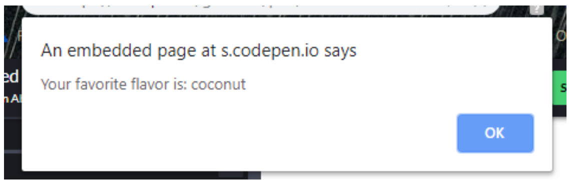
(continues on next page)

(continued from previous page)

```
        handleSubmit(event)
            {
                alert('Your favorite flavor is: ' + this.state.value);
                event.preventDefault();
            }

        render()
        {
            return (
            <form onSubmit={this.handleSubmit}>
            <label>
            Pick your favorite flavor:
            <select value={this.state.value} onChange={this.handleChange}>
                <option value="grapefruit">Grapefruit</option>
                <option value="lime">Lime</option>
                <option value="coconut">Coconut</option>
                <option value="mango">Mango</option>
            </select>
            </label>
            <input type="submit" value="Submit" />
            </form>
            );
        }
    }

ReactDOM.render(
<FlavorForm />,
document.getElementById('root')
);
```

For this form you can see that the user is able to make a choice of what flavor they would like from the drop-down list. Once the user selects a flavor, a message is printed to the user interface. This message lets the user know the flavor they selected.


Pick your favorite flavor: Coconut ▼ Submit


An embedded page at s.codepen.io says

Your favorite flavor is: coconut

OK

3.17.5 Reception of ReactJS

Since ReactJS was opened to the public, it gained some pretty interesting attention a couple years after its release. According to [InfoQ] ReactJS was receiving some mixed reviews from the development community. One of the criticisms the community had was how the JavaScript library made it difficult to separate the HTML code as the front-end language and the JavaScript code as the back-end language. Usually designers would work separately on these two coding languages and could hook them together. However, with ReactJS you are able to morph the two languages together, which made it tricky at first for developers to understand.

Another criticism that faced ReactJS came from its terms and conditions clause right after its initial launch. According to [MediumArticle], Facebook stated that if you were to agree to its terms and conditions, you did not have the right to sue Facebook or its subsidiaries, but Facebook and its subsidiaries could sue the user if they believed that someone was infringing on their patent rights. This particular clause was vague and didn't sit well with developers. Many developers and companies thought that investing in ReactJS would be too much of a risk for them, that their industries could be hurt if they built with ReactJS.

As people started to distance themselves from the JavaScript library, Facebook came out and said that they would change their patent clause. According to [FacebookCode], the ReactJS development team wanted to make sure that anyone who used their JavaScript library felt confident using it. The development team stated that it wouldn't sue industries for patent infringement with the vague terms that were used. Facebook cleared up the confusing statement by stating clear examples of that patent infringement looked like. Cleaning up this patent clause was intended to help programmers and companies feel comfortable with using the JavaScript library again, without the fear of worrying about legal action.

3.17.6 Conclusion

When it comes to ReactJS, it is apparent that this JavaScript library is one that has the potential to create simple, yet stunning user interfaces. The syntax that comes with this library is easy to understand once you grasp how all the unique features of the language interact with one another. This particular JavaScript library took both HTML and JavaScript code and morphed the two languages into one. This was seen as an interesting step in the development community, while others saw it as something as too complex. At first, industries were afraid to work with ReactJS due to its vague patent infringement clause. But as Facebook and the ReactJS Development team could see the backlash they were receiving, they changed their clause so companies could feel comfortable with using their JavaScript library. Even though ReactJS has had its problems, it is still an up and coming JavaScript library that has the potential to create sophisticated user interfaces for developers around the world.

3.17.7 Citations

Bibliography

[Arai] Arai. "Intl" Intl, MDN. 05 Apr. 2017. Web. 13 Apr. 2017

[Block_scope] "Javascript: Block scope." Programmer and Software Interview Questions and Answers. Programmer-Interview, n.d. Web. 06 Apr. 2017.

[Compatibility] "ECMAScript 6 compatibility table" ECMAScript 6 compatibility table. kangax., 2016. Web. 04 Apr. 2017.

[ECMAScript_6] Engelschall, Ralf S. "ECMAScript 6: New Features: Overview and Comparison" ECMAScript 6: New Features: Overview and Comparison. Ralf S. Engelschall, 2017. Web. 04 Apr. 2017.

[es6_Features] Hoban, Luke. "Lukehoban/es6features" GitHub. N.p., 24 July 2016. Web. 04 Apr. 2017

[PR_Newswire] PR Newswire. "Lounge Lizard Highlights 3 Ways to Improve JavaScript with ES6." PR Newswire US. PR Newswire, 03 June 2016. Web. 4 Apr. 2017

[Prusty] Prusty, Narayan. Learning ECMAScript 6: learn all the new ES6 features and be among the most prominent JavaScript developers who can write efficient JS programs as per the latest standards! Birmingham: Packt Publishing, 2015. Print.

[Simpson] Simpson, Kyle. You Don't Know JS: ES6 & Beyond. Sebastopol: O'Reilly Media, 2016. Print.

[Zakas_Understanding] Zakas, Nicholas C. Understanding ECMAScript 6: The Definitive Guide for Javascript Developers. San Francisco: No starch Press, 2016. Print.

[PurdueMLA] "MLA In-Text Citations: The Basics" Purdue University. Purdie Online Writing Lab, 3/27/2019.

[TextCitation] Author's Last name, First name. "Title of Source." Title of Container, other contributors, version, numbers, publisher, publication date, location.

[dta] "Directory traversal attack." Wikipedia. Wikimedia Foundation, 07 Feb. 2017. Web. 15 Feb. 2017.

[sphinx] Georg Brandl. "reStructuredText Primer" Sphinx Team, Web. 15 Feb. 2017.

[freeclassifieds] Chris White. "Pembroke Welsh Corgi Puppies Available" Free Classifieds USA, Web. 14 Jul. 2018.

[React] "React: A JavaScript library for building user interfaces" Facebook Inc. Facebook Open Source, Web 2 April. 2019.

[w3React] "What is React? " W3 Schools. Refnes Data, Web 4 April. 2019.

[learnReact] "Borgen, Per Harald" freeCodeCamp.org. A Medium Corparation. 4/10/18.

[reactTutorial] "McGinnis Tyler" TylerMcGinnis.com, np. March, 12. 2018._

[futureReact] “Caliman, Diana” 2019 Creative Tim, Creative Tim’s Blog. April 13,2018._

[TicTacReact] “Dan Abramov” Facebook Inc. Facebook Open Source, Web 16 April. 2019.

[reactBackground] “Dawson, Chris” 2019 The New Stack, The New Stack.

[reactIntro] “Tutorial: Intro to React” React. Facebook Inc., 4/2/2019.

[reactSimple] Borgen, Per Harald. “Learn React.js in 5 Minutes” FreeCodeCamp, A Medium Corporation, 4/10/2018.

[reactHistory] Papp, Andrea. “The History of React.js on a Timeline” RisingStack, RisingStack Inc., 7/20/2018.

[reactW3Schools] “What is React?” W3Schools, 4/3/2019.

[reactPopularity] Kostrzewa, Denis. “Is React.js the Best Javascript Framework in 2018?” Hacker Noon, A Medium Corporation, 7/19/2018.

[reactBestPractices] “ReactJS Best Practices.” Tutorials Point, 4/4/2019.

[VueIntroduction] “Introduction: What is Vue.js?” Vue.js. Web. 2 Mar. 2019.

[VueWiki] “Vue.js” Wikipedia. Wikimedia Foundation, Web. 4 Apr. 2019.

[w3schoolsVue] “What is Vue.js?” w3schools. Refsnes Data, Web. 4 Apr. 2019.

[Egghead] “Evan You, creator of Vue.js” Egghead.io. Egghead.io, Web. 9 Apr. 2019.

[Nodejs] Node.js Foundation. “Node.js” Joyent Inc, Web. 2 April. 2019.

[NodejsDev] Node.js Foundation. “Node.js Dev” Joyent Inc, Web. 2 April. 2019.

[LearningNode] Powers, Shelly, *Learning Node*. Sebastopol, O’Reilly, 2015.

[EventLoop] Node.js Foundation. “Node.js Event Loop” Joyent Inc, Web. 19 April. 2019.

[SocketIo] Damien Arrachequesne. “Socket.io” Socket.io, Web. 29 April. 2019.

[FullStackReact] Lerner, Ari “30 Days of React: What is React?” Fullstack React, 2017. Web. 2 April 2019.

[ReactHistory] Dawson, Chris “Javascript’s History and How it led to ReactJS” The New Stack, 25 July 2014. Web. 4 April 2019.

[Hackernoon] Kostrzewa, Denis “Is React.js the Best Javascript Framework in 2018?” Hacker Noon. Hacker Noon, 19 July 2018. Web. 8 April 2019.

[Medium] Mahmood, Hamza “Advantages of Developing Modern Web apps with React.js” Medium. Medium, 27 May 2018. Web. 15 April 2019.

[Neuhaus] Neuhaus, Jens “Angular vs. React vs. Vue: A 2017 Comparison” Medium. Medium, 28 August 2017. Web. 20 April 2019.

[Timeline] Papp, Andrea “The History of React.js on a Timeline” Rising Stack. Rising Stack, 4 April 2018. Web. 20 April 2019.

[Bostock1] Bostock, Mike. “Data-Driven Documents.” D3.Js.

[Bostock2] Bostock, Mike. “How Selections Work.” 26 Apr. 2013.

[Bostock3] Bostock, Mike. “OMG Particles!” Popular Blocks, 20 Feb. 2019.

[Murray] Murray, Scott, et al. “Data Driven Documents.” VisWeek 2012, 2012.

[StackShare] “Why Developers like D3.Js.” StackShare, StackShare Inc.

[IntroNodeJS] Node.JS Intro “Introduction” Google, Web 4/2/2019

[Node.jsIntroduction] Node.JS Introduction “Introductions” Google, Web 4/4/2019

[NodeJSWebApp] Node.JS Advantages “Advantages and disadvantages” Google,Web 4/4/2019

[WhyUseNodeJS] Why use Node.JS "Why use NodeJS" Google, Web 4/4/2019

[NodeJSTutorials] Node.JS Tutorials "Tutorials" Google, Web 4/4/2019

[BeforeNodeJS] Before Node.JS "Before NodeJS" Google, Web 4/4/2019

[SingleThreadMechanism] Node.JS Code "Single Thread" Google, Web 4/4/2019

[w3SchoolsRef] "HTML Responsive Web Design" w3Schools. w3Schools.com, 4/4/2019.

[diviSpace] John Anderson. "How to use Stylebot:" divi.space, Web. 18 Dec. 2017.

[MediaQueries] "Media Queries" w3Schools. w3Schools.com, 4/16/2019.

[SmashingMagazine] Christian Krammer. "How To Setup A Print Style Sheet" SmashingMagazine. smashingmagazine.com, 4/16/2019.

[CSSHistory] Bert Bos. "History on CSS" Style Activity Lead, Web. 17 Dec. 2016.

[w3SchoolsMediaQueries] "More on Media Queries" w3Schools. w3Schools.com, 4/17/2019.

[AMP] AMP "Accelerated Mobile Pages Project.."

[AMPWiki] Wikipedia "Accelerated Mobile Pages.." Wikimedia Foundation

[rjs] "Reactjs website." Facbook Open Source, Facebook Inc, 04/09/2019. Web. ND.

[built_sitessites] "Top 32 Sites Buit with ReactJS." Medium, Coder Academy, 04/09/2019. Web. 06/09/2016.

[rjs_historyhistory] "The History of React.js on a Timeline." RisingStack, Andrea Papp, 04/09/2019. Web. 04/04/2018.

[use_ractjs] "What Is ReactJS and Why Should We Use It?." C-sharpcorner, Nitin Pandit, 04/09/2019. Web. 11/14/2018.

[versus] "Comparison with Other Frameworks." Vue.js, Vue.js, 04/09/2019. Web. ND.

[not_use_reactjs] "More Than React: Why You Shouldn't Use ReactJS for Complex Interactive Front-End Projects, part 1." InfoQ, Yang Bo, 04/09/2019. Web. 01/30/2017.

[reactjs_classification] "Is React a library or framework and why? <https://www.quora.com/Is-React-a-library-or-a-framework-and-why>'_." Quora, Brian Engelhardt, 11/13/2017, 04/18/2019

[WhatIsVue] "Introduction - Vue.js." Vue.js, n.d. Web. 11 Apr. 2019.

[HistoryOfVue] Evan You "First Week of Launching Vue"Blog, 11 Feb. 2014, Web. 28. Apr. 2019

[Installation] "VueJS Environment Setup.", Tutorials Point, n.d. Web. 4 Apr. 2019.

[Popularity] Nowak, Maja. "Reasons Why Vue.js Is Getting More Traction Every Month.", Monterail, 19 Dec. 2018, Web. 28 Apr. 2019.

[Tutorial] Eschweiler, Sebastian. "Vue.js 2 Quickstart Tutorial 2017.", CodingTheSmartWay, Medium, 7 Jan. 2017, Web. 11 Apr. 2019.

[Buna] Samer Buna "Requiring modules in Node.js: Everything you need to know" Freecodecamp, Web. 19 Mar, 2017

[Thinkmobiles] "Why use Node.js - look behind the scenes of web development" Thinkmobiles, Web. 04 Apr, 2019

[Tutorialspoint] "Node.js Tutorial" Tutorialspoint, Web. 02 Apr. 2019

[W3Schools] "Node.js MySQL" W3Schools, Web. 18 Apr. 2019

[w3schools] "AngularJS Tutorial" W3Schools, Web. 4 Apr. 2019

[Lau] "sitepoint" sitepoint, 05 Sept. 2013. Web. 4 Apr. 2019

[Ray] “Why AngularJS is my preferred framework for software development” freeCodeCamp, 16 Jul. 2018. Web. 4 Apr. 2019

[Huszárik] “AngularJS to Angular” RisingStack, Web. 4 Apr. 2019

[Github] “GitHub - mrdoob/three.js: JavaScript 3D library”, r103, Ricardo Cabello, Web 2 April, 2019.

[Mozilla] “WebGL: 2D and 3D graphics for the web”, Scholz, Florian, Mozilla, Web 2 April, 2019.

[Threejs] “three.js - Javascript 3D library” Cabello, Ricardo, Web 2 April, 2019.

[React_OS] “React, A Javascript library for building user interfaces.” reactjs, Facebook Open Source, 2019

[React_hackernoon] “React.js: a better introduction to the most powerful UI library ever created.” Medium, 03 Sep. 2018

[RisingStackScott] “The History of React.js on a Timeline.” RisingStack, March. 2018

[TheNewStackScott] “JavaScript’s History and How it Led To ReactJS.” TheNewStack, 25 Jul. 2014

[React_FutureScott] “10 Famous Apps Using ReactJS Nowadays.” Brainhub.

[React_code_tutScott] “Learn React.js in 5 minutes.” freecodecamp, 10 Apr. 2018

[Alicea] Anthony, Alicea “Master AngularJS (Essential JavaScript Concepts) ” Udemy, Anthony Alicea, Web 4/9/2019

[Angular] Angular.io “Architecture overview “version 7.2.12-local+sha.d727561, Google, Web 4/2/2019

[TutorialspointAngular] Tutorialspoint.com “AngularJS - Overview “Web 4/2/2019

[TutorialspointAngularMVC] Tutorialspoint.com “MVC Framework - Introduction ” Web 4/4/2019

[W3SchoolsAngular] W3schools.com “AngularJS Routing” Web 4/16/2019

[Austin] Andrew Austin “An Overview of AngularJS for Managers. ” Andrew Austin, 14 Aug. 2014

[Strahl] Strahl, Rick. “AngularJs and Promises with the $Http Service. ” Rick Strahl’s Web Log, Rick Strahl, Web 4/10/2019

[Rajput] Rajput, Mehul “The Pros and Cons of Choosing AngularJS. ” JAXenter, 21 Mar. 2016

[ReactJSTutorial] “Tutorial: Intro to React” React. Facebook Inc, Web. 4 Apr. 2019.

[ReactJSComponents] “Components and Props in ReactJS” React. Facebook Inc, Web. 4 Apr. 2019.

[ReactJSHandling] “Handling Events in ReactJS” React. Facebook Inc, Web. 4 Apr. 2019.

[ReactJSForms] “Forms in ReactJS” JSX, Facebook Inc, Web. 4 Apr. 2019.

[InfoWorld] Krill, Paul. “React: Making Faster, Smoother UIs for data-driven Web Apps” InfoWorld Tech Watch, InfoWorld, Web. 15 May 2014.

[TheNewStackArticle] Dawson, Chris. “JavaScript’s History and How it Led to React JS” The New Stack Technology, The New Stack, Web. 25 Jul. 2014.

[InfoQ] Hemel, Zef. “Facebook’s React JavaScript User Interfaces Library Receives Mixed Reviews” InfoQ News, InfoQ, Web. 3 Jun. 2013.

[MediumArticle] Berkana. “A Compelling Reason Not to Use ReactJS” Bits and Pixels, A Medium Corporation, Web. 24 May 2015.

[FacebookCode] Pearce, James. “Updating Our Open Source Patent Grant” Facebook Code, Facebook Inc, Web. 10 Apr. 2015.