Web Applications

CSE183

Fall 2020

Assignment 3 - JavaScript I

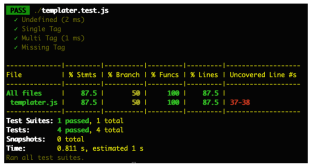
Baskin Engineering
UC SANTA CRUZ

Assignment 3 - JavaScript I

- Simple JavaScript class for replacing tags in a template
  - Template: `'Hello {{tag}}'`
  - Map: `{tag: 'World'}` (a JS object - a.k.a. a 'dictionary')
  - Result: `'Hello World'`
- Need to install **Node.js** and **npm** (Node Package Manager)
  - Follow instructions in assignment specification
  - Seek help ASAP (i.e. today) if you run into installation problems
- Learning outcomes:
  - Basic familiarity with JavaScript
  - Exposure to testing frameworks
  - Experience of 'linters' (code quality checkers)
- WARNING: Don't be tempted to use a 3rd party component ☹
  - Write it yourself from first principles ☺

Assignment 3 - JavaScript I

- Basic:
  - Implement a basic `Templater` class
  - Pass tests given to you
    - `npm run test`
- Advanced:
  - Add new tests for advanced features
  - Modify `Templater` class to pass them
- Stretch:
  - Remove all 'linter' warnings
    - `npm run lint`
  - Demonstrate 100% code coverage



Upcoming Lectures

- Monday: JavaScript III & Document Object Model
- Wednesday: JavaScript Wrap-up
- Friday: User Interfaces

Tasks

- **Assignment 3** due 23:59 **Thursday October 22**

Web Applications

CSE183

Fall 2020

JavaScript III

Baskin Engineering UC SANTA CRUZ

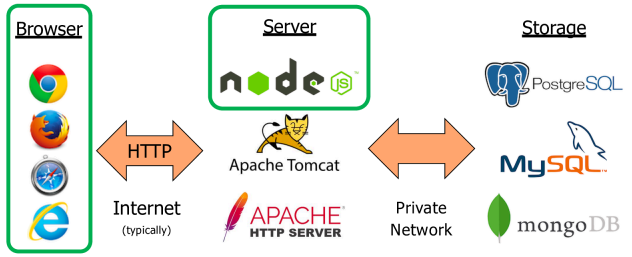
Notices

- **Assignment 3** due 23:59 **Thursday October 22**

Today's Lecture

- JavaScript Recap
- ECMAScript Review
- ECMAScript New Features
- Quiz 1 Solutions
- Questions

Full Stack Web Applications



JavaScript - Overview

- High-level
  - Heavily abstracted from hardware details
- Interpreted
  - Not compiled, executed by a platform-dependent run-time environment
- Dynamic
  - Undertakes compiler-like operations at runtime
- Untyped / Dynamically Typed
  - Any variable can hold any type of data
- Prototype-based
  - Object-oriented behaviors are re-used (inherited) from existing objects (prototypes)
- Has first-class functions
  - Functions are objects and can be manipulated as such
- Programming Models
  - Imperative, Functional, Object-oriented

ECMAScript

- Standardised Version of JavaScript
  - Updated every 12 months
  - Straightforward process for getting new features into the language
- Transpilers:
  - Write in up-to-date style but allow execution by any interpreter
    - i.e. modern JS in, old JS out
  - E.g. Babel https://babeljs.io/

    Given `var name = 'Bob';`
    then `console.log(`Hello ${name}`);`
    becomes `console.log('Hello ' + name);`
    or `console.log('Hello '.concat(name));`
- Front-end Frameworks embrace new language features
  - Reacj.js - Uses modern ECMAScript features
  - Angular - Uses TypeScript (JavaScript with static type checking)

Some ECMAScript Features

- Already encountered:
  - `let`, `const`, `class`
- Additional common features:
  - Default Parameters
  - Rest Parameter
  - Spread Operator
  - Destructuring Assignment
  - Template Literals
  - For Loops
  - Modules
  - Collection Abstractions
  - Asynchronous Operations

Default Parameters

- Named parameters initialized with default values
- Old way:

```
function oldWay(a,b) {
  a = a || 2;
  b = b || 'Hello';
  return b + a;
}
```

- New way:

```
function newWay(a = 2, b = 'Hello') {
  return b + a;
}
```

- Same output from:

```
console.log(oldWay());
console.log(newWay());
console.log(oldWay(undefined));
console.log(newWay(undefined));
```

Rest Parameter …

- Represent indefinite number of arguments as named array
- Old way:
  - No parameters listed, but available as `arguments` array

```
function oldWay() {
  var a = arguments[0];
  var b = arguments[1];
  return a + b;
}
```

- New way:
  - Additional parameters placed in named array

```
function newWay(a,...argv) {
  var b = argv[0];
  return a + b;
}
```

- Same output from:

```
console.log(oldWay(1,2));
console.log(newWay(1,2));
```


Spread Operator …

- Expand iterate-able expressions (arrays, strings) into argument lists or elements
- Given:

```
function sum(x, y, z) {
  return x + y + z;
}
const numbers = [1, 2, 3];
```

- Old way:

```
var s = sum.apply(null, numbers);
```

- New way:

```
let s = sum(...numbers);
```

- Same output from:

```
console.log(s);
```


Destructuring Assignment

- Unpack values from arrays, or properties from objects, into distinct named variables
- Examples:

```
var a = arr[0];
var b = arr[1];
var c = arr[2];
```

```
let [a,b,c] = arr;
```

```
var fname = obj.fname;
var lname = obj.lname;
bar title = obj.title
```

```
let {fname, lname, title} = obj;
```

```
function oldWay(person) {
  var fname = person.fname;
  var age = person.age;
  return fname + age;
}
```

```
function newWay({fname, age}) {
  return fname + age;
}
```

- Same output from:

```
console.log(oldWay({fname: 'Bob', age: 24}));
console.log(newWay({fname: 'Bob', age: 24}));
```


Template Literals

- String literals supporting embedded expressions
- Given:

```
var theirName = 'Bob';
var yourName = 'Patsy';
```

- Old way:

```
var str = 'Hello ' + theirName +', my name is ' + yourName + '.';
```

- New way:

```
let str = `Hello ${theirName}, my name is ${yourName}`;
```

- Same output from:

```
console.log(str);
```

For `of`

- Loop over iterate-able objects
  - E.g. String, Array, Map, Set, …
- Given:

```
var arr = [5,6,7];
var sum = 0;
```

- Old way:

```
for (var i = 0; i < arr.length; i++) {
  sum += arr[i];
}
```

- New way:

```
for (elem of arr) {
  sum += elem;
}
```

- Same output from:

```
console.log(sum);
```


For `in`

- Loop over string-keyed object properties
- Given:

```
var obj = { a: 5, b: 6, c: 7 };
var sum = 0;
```

- Old way:

```
var keys = Object.keys(obj);
for (var i = 0; i < keys.length; i++) {
  sum += obj[keys[i]];
}
```

- New way:

```
for (prop in obj) {
  sum += obj[prop];
}
```

- Same output from:

```
console.log(sum);
```


Modules

- Visibility of variables within source file
  - When "including" one file in another, need a way to restrict access to "private" variables in the included file
- Old way:

```
var exportedName = (function () {
  var x, y, x;   // 'Private' variables
  ...
  return {x: x, y: y};
})();
```

- New way:

```
var x, y, x;     // still private because they're not exported
...
var exportedName = {x: x, y: y};
export exportedName;
```

- Browser: `export` / `import`
- Node.js: `module.exports` / `require()`

Additional Features

- Collection Abstractions
  - `Set`, `Map`, `WeakSet`, `WeakMap`
  - Self study for details
- Asynchronous Operations
  - `async` / `await` and `Promise`
  - We'll cover later in the class

Quiz 1 - Solutions

Select the technology tiers that typically appear in a Full Stack Web Applications with a multitiered architecture.

Browser

Server

Data Storage

Experienced Web App developers consider using the HTML tags <b> and <i> to be bad practice. Explain why this is a valid opinion.

Tags request styling of text (bold/italic) yet it is considered *good* practice to enforce **separation of content** (HTML) **from style** (CSS).

Can be replaced by the semantic and tags

If a CSS Style Sheet does not explicitly set the width and height of an element, how does the browser decide what size to render the element?

Depends on the element type

Defaults to width:auto

Depends on the browser

JavaScript allows var statements to be written anywhere in a function, yet some coding standards require they be placed at the start of functions. Give a reasoned argument as to why this requirement is considered valid.

Var declarations are **hoisted** to the start of functions by the JavaScript runtime regardless of where they are declared in the code.

Insisting they be placed at the start of the function is reasonable as the code as written matches the code as executed and simultaneously avoids confusion and the potential for bugs.

If the current location of a browser tab is http://www.example.com/sales/index.html which contains the links:

Current Stock

Texas Rollup

What are the full URL equivalents of these links?

http://www.example.com/stock/current/index.html (absolute)

http://www.example.com/sales/rollup/texas/index.html (relative)

What would this snippet of JavaScript display on the console? Explain your answer using line numbers for reference.

```
5  let i = 4;
6  (function(i){
7    i /= 6.2;
8    console.log(i);
9  })(12.4);
10 console.log(i);
```

Console output:

2
4

The **closure** of the function includes the argument i but not the global variable i

Line 8 displays the result of dividing the argument i by 6.2

Line 10 displays the value of the global i

What port would the server this URL is hosted by have to be listening on for a browser to connect to it successfully?

https://www.example.com/sales/index.html

443

Is this snippet of JavaScript valid, and if so, what would it display on the console? Explain your answers.

```
11  let arr = ['one','two',,,5,6];
12  console.log([arr[5],arr[3],arr[6]]);
```

Console output: [6, undefined, undefined]

JavaScript arrays are **polymorphic** and **sparse**; hence the code is valid even though the array has missing elements and elements of two different types, namely number and string.

As JavaScript arrays are zero indexed, arr[5] is the number 6, the 6th element in the array.

There is no element at index 3 but JavaScript arrays are sparse so that element has a value of "undefined".

There is no element at index 6 as the length of the array is 6, but again, JavaScript arrays are sparse, so any non-negative index is valid but in this case the value is also "undefined".

Select the statements that are true about fixed and absolute CSS positions.

fixed positions the element with respect to the page

absolute positions the element with respect to its closest ancestor

fixed positioning means the element is fixed in place as scrolling occurs

Describe how invalid characters are dealt with in query parameters and give an example.

Query parameters must be **URL Encoded**. i.e. each invalid character is replaced by %xx where xx is the hexadecimal value of the character.

Example:

https://www.demo.com/customer?name=M&S Stores

Becomes:

https://www.demo.com/customer?name=M%26S%20Stores

Upcoming Lectures

- Wednesday: Document Object Model
- Friday: User Interfaces

- Sometime this week: Assignment 2 Feedback

Tasks

- **Assignment 3** due 23:59 **Thursday October 22**

Web Applications

CSE183

Fall 2020

Document Object Model

Baskin Engineering UC SANTA CRUZ

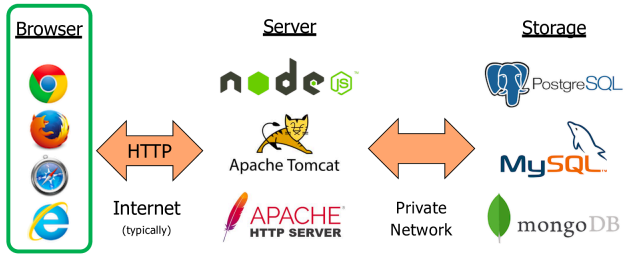
Notices

- **Assignment 3** due 23:59 **Thursday October 22**
- **Assignment 1** grades posted

Today's Lecture

- Document Object Model (DOM)
  - Hierarchy
  - Node Properties, Methods, Mutators
  - DOM and CSS
  - Coordinates & Dimensions
- DOM Events
  - Handling
  - Event Objects
  - Precedence
  - Timer Events
- Questions

Full Stack Web Applications



Document Object Model

Document Object Model (DOM)

- HTML **document** exposed as a collection of JavaScript **objects**
- JavaScript can query and modify the HTML document via DOM
- Accessed via the JavaScript global scope:
  `window` when using `'use strict'`
  `this` otherwise

DOM Hierarchy

- Starts at `window.document`
  - Can be shortened to simply `document`
- Follows HTML structure:
  `document.head`
  `document.body`
- DOM Objects have *hundreds* of properties
  - Though most are private
- DOM Objects have a common set of properties and methods known as the DOM "Node"

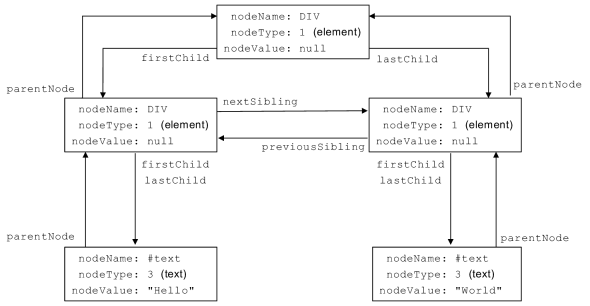
The DOM Node

- Properties
  `nodeName`
  - Element tag (P, DIV, etc.) always in UPPER CASE

  `parentNode`, `nextSibling`, `previousSibling`, `firstChild`, `lastChild`
  - Codification of the document hierarchy
- Methods
  `getAttribute`, `setAttribute`, ...
  - Accessors and mutators

DOM Nodes

```
<div>
  <div>Hello</div>
  <div>World</div>
</div>
```



https://www.w3schools.com/jsref/prop_node_nodetype.asp

Finding DOM Nodes

- Walk the hierarchy (not recommended)

```
let element = document.body.firstChild.nextSibling;
```

- Use a lookup method
  - Several of these Self Study
  - E.g. by element id

```
<div id="mydiv">...</div>
let myDiv = document.getElementById('mydiv');
```

  - E.g. by tag name

```
let allDivs = document.getElementsByTagName('div');
```


Common Properties & Methods

`textContent` All the text in node and its descendants

```
document.getElementById('container').textContent
```

is `'Hello World'`

`innerHTML` HTML of node's descendants

```
document.getElementById('container').innerHTML
```

is `'<div>Hello</div><div>World</div>'`

`outerHTML` HTML of node and its descendants

```
document.getElementById('container').outerHTML
```

is `'<div id="container"><div>Hello</div><div>World</div>'`

`getAttribute()` / `setAttribute()`

get and set attributes of an element

```
<div id='container'>
  <div>Hello</div>
  <div>World</div>
</div>
```


Common Mutators

- Change the content of an element

```
element.innerHTML = '<p>A Paragraph</p><div>A DIV</div>'
```

  - Replaces content but retains attributes

- Change the `src` attribute of an image tag

```
img.src = 'foo.png'
```

- Set visibility of elements

```
element.style.display = "none";          // not visible and uses no space
element.style.visibility = "hidden";     // invisible, but uses space
```


DOM and CSS

- Change an element's class (i.e. change its style completely)

```
element.className = 'active';
```

- Update an element's style (not recommended approach)

```
element.style.color = '#ff000';
```

- Query DOM using a CSS selector

```
document.querySelector()
document.querySelectorAll()
```

Details? Self Study

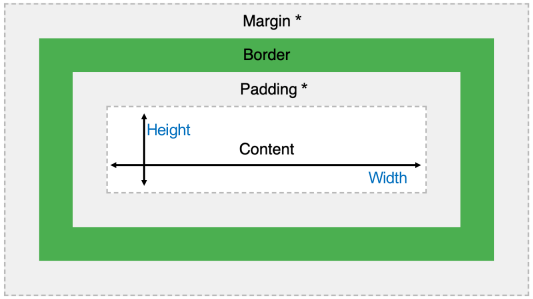
Modifying the DOM Hierarchy

- Create a new element
```
element = document.createElement('DIV');
```
- Clone an existing element
```
element = document.getElementById('myDiv').cloneNode();
```
- Add to an existing Node
```
parent.appendChild(element);
parent.insertBefore(element, sibling);
```
- Remove Nodes
```
parent.removeChild(element);
```
- Yet most developers find setting `innerHTML` easier
  - Yet this is, arguably, poor practice

Miscellaneous DOM Operations

- Redirect to a new page
```
window.location.href = "newPage.html";
```
  - Warning: interrupts JavaScript excecution
- Communicate with the user
```
alert('Hello World!');
confirm('Are you quite sure?');
```
- Developer messages
```
console.log()
console.debug()
console.error()
```
  - And many more Self Study

The CSS Box Model



* Transparent

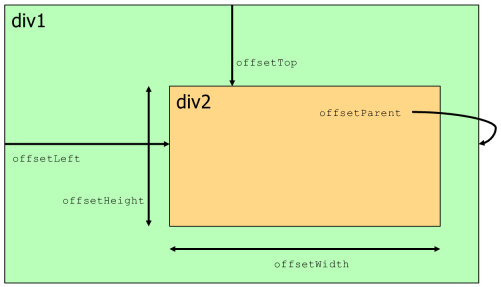
https://www.w3schools.com/css/css_boxmodel.asp

DOM Coordinates

- Top left corner is the origin (0,0)
- Element position defined by the top left corner of its **margin**
- Read with:
```
element.offsetLeft
element.offsetTop
```
- Coordinates are relative to:
```
element.offsetParent
```
  - Not necessarily the same as `element.parentNode`

DOM Coordinates

```
<div class='div1'><div class='div2'></div></div>
```



Element Dimensions

- Reading:
  `element.offsetWidth` and `element.offsetHeight`
  - Includes contents, padding, border but not margin
- Updating:
  `element.style.width` and `element.style.height`

DOM Events

- DOM "talks to" JavaScript via Events
  - JavaScript manipulates DOM via methods and mutators
- Principal Event Types:
  - Mouse (movement, button clicks, enter / leave an element)
  - Keyboard (key up, down, pressed)
  - Focus
  - Input Field Traversal
  - Timer
- Other Event Types:
  - Element content changed
  - Page loaded or unloaded
  - Image loaded
  - Uncaught exceptions
  - And many more… Self Study

Event Handling

- Create an **Event Handler**
- Specify:
  - The event of interest (what happened)
  - The element of interest (which element it happened to)
  - JavaScript to execute when event happens to the element
- Options:
  - In HTML:

```
<div onclick = "alert('Hello World');">Hello World</div>
```

element event JavaScript

  - In JavaScript via DOM

```
let mouseClick = function() { alert('Hello World'); }
element.onclick = mouseClick;
```

Or `element.addEventListener('click', mouseClick);`

The Event Object

- Event listener functions passed an `Event` object
- Typically sub-classed `MouseEvent`, `KeyboardEvent`, etc.
- Selected `Event` properties:
  - `type` name of the event (`'click'`, `'mouseDown'`, `'keyUp'`, …)
  - `timeStamp` when the event was created
  - `currentTarget` element listener was registered on
  - `target` element that dispatched the event

Mouse and Keyboard Events

- Selected `MouseEvent` properties (prototype inherits from `Event`)
  - `button` mouse button that was pressed
  - `pageX`, `pageY` mouse position relative to the page origin
  - `screenX`, `screenY` mouse position in screen coordinates
- Selected `KeyboardEvent` properties (prototype inherits from `Event`)
  - `keyCode` identifier for the keyboard key that was pressed
    - Not necessarily an ASCII character
  - `charCode` integer Unicode value corresponding to keypress
    - But only if there is one

Draggable Circle

HTML

```
<body>
  <div class="draggable"
    onmousedown="mouseDown(event);"
    onmousemove="mouseMove(event);"
    onmouseup="stopDragging(event);"
    onmouseleave="stopDragging(event);">
    Drag Me!
  </div>
</body>
```

CSS

```
.draggable{
  position: absolute;
  height: 200px;
  width: 200px;
  background-color: yellow;
  display: flex;
  justify-content: center;
  align-items: center;
  border: 3px solid green;
  border-radius: 100px;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 30px;
}
```

JavaScript

```
let dragging = false;
let prevX, prevY;

function mouseDown(event) {
  prevX = event.pageX;
  prevY = event.pageY;
  dragging = true;
  event.target.style.backgroundColor = 'green';
}

function stopDragging(event) {
  dragging = false;
  event.target.style.backgroundColor = '';
}

function mouseMove(event) {
  if (!dragging) {
    return;
  }
  let elem = event.target;
  elem.style.left = (elem.offsetLeft +
    (event.pageX - prevX)) + "px";
  elem.style.top = (elem.offsetTop +
    (event.pageY - prevY)) + "px";
  prevX = event.pageX;
  prevY = event.pageY;
}
```


Event Precedence

- Complication ☹
  - Elements can overlap with others
  - Suppose user clicks their mouse on "Hello World"

```
<body>
  <table>
    <tr>
      <td>Hello World</td>
    </tr>
  </table>
</body>
```

  - If we have event handlers on the `table`, `tr` and `td` elements, which handler gets called?
    - Self Study (find out by writing the code, use "Draggable Circle" as a guide)
  - Sometimes more convenient for outer elements to handle events
  - Sometimes more convenient for inner elements to handle event

Event Capture and Bubbling

- Capture phase (a.k.a. "trickle-down")
  - Start at the *outermost* element and work down to the innermost
  - Each element can stop the capture, so children never see the event:
    `event.stopPropagation()`
- Bubble phase (a.k.a. "bubble-up")
  - Invoke handlers on the *innermost* element that dispatches the event
    - Usually the most appropriate thing to do
  - Repeat on parent, grandparent, etc.
  - Any element can stop the bubbling:
    `event.stopPropagation()`
  - Can insist on Bubble only:
    `element.addEventListener(eventType, handler, false);`
    `useCapture` flag
  - Handlers in bubble phase more common than capture phase

Timer Events

- Used for animations, page refreshes, forced logout etc.
- Given
  `function foo() { … }`
- Run `foo` once, 5 seconds from now
  `let token = setTimeout(foo, 5*1000);`
- Run `foo` every 50 milliseconds
  `let token = setInterval(foo, 50);`
- Cancel a timer
  `clearInterval(token);`

Event Concurrency

- Events are serialized and processed one-by-one
- Event handling *does not* interleave with other JavaScript
  - Handlers run to completion
  - Not preemptable
  - Not multi-threaded
- Make reasoning about concurrency easier
  - Rarely need locks
- Background processing is much harder than with threads

Event Based Programming

- Must wait for something to invoke your code
- Must return quickly from the handler
  - Otherwise Web App becomes unresponsive
- Key is to maintain control through events
  - Make sure you have declared enough handlers
  - Timers should be used only as a last resort
- Node.js provides an event dispatching mechanism for for server-side JavaScript programming
  - We'll see this later in the class

Upcoming Lectures

- Friday: User Interfaces
  - Plus Assignment 2 Feedback & Assignment 4 Overview
- Monday: Introduction to React
- Wednesday: Single Page Web Applications
- Friday: Quiz 2

Tasks

- **Assignment 3** due 23:59 **Thursday October 22**

Web Applications

CSE183

Fall 2020

User Interfaces

Baskin Engineering
UC SANTA CRUZ

Notices

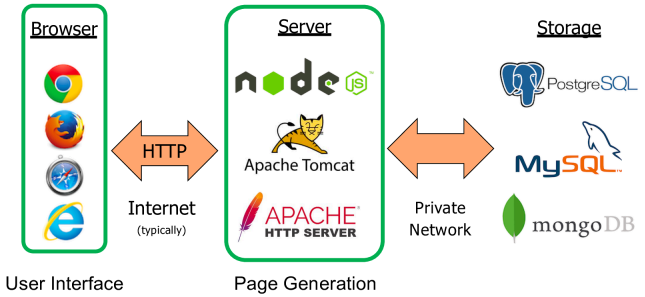
- **Assignment 4** due 23:59 **Thursday October 29**
- **Quiz 2** during class **Friday October 30**

Today's Lecture

- DOM Events Revisited
- Event Concurrency
- 1st, 2nd, 3rd Generation Web Application Frameworks
- Templating
- Model View Controller
- 4th Generation Web Application Frameworks

- Assignments 2 & 3 Review
- Assignment 4 Introduction
- Questions

Full Stack Web Applications



DOM Events Revisited

- Handlers

```
<div onclick = "alert('Hello World');">Hello World</div>
```

element event JavaScript

Or

```
let mouseClick = function(event) { alert('Hello World'); }
element.addEventListener('click', mouseClick);
```

- The `Event` object and descendants `MouseEvent`, `KeyboardEvent`, etc.
- Precedence
- Capture and Bubbling
- Timer Events

Event Concurrency

- Events are serialized and processed one-by-one
- Event handling *does not* interleave with other JavaScript
  - Handlers run to completion
  - Not preemptable
  - Not multi-threaded
- Makes reasoning about concurrency easier ☺
  - Rarely need locks
- Background processing is much harder than with threads ☹

Event Based Programming

- Must wait for something to invoke your code
- Must return quickly from the handler
  - Otherwise Web App becomes unresponsive
- Key is to maintain control through events
  - Make sure you have declared enough handlers
  - Timers should be used only as a last resort
- Node.js provides an event dispatching mechanism for server-side JavaScript programming
  - We'll see this later in the class

Web App History Revisited

- Initially static HTML files with HTML forms for input
- Then Common Gateway Interface (CGI)
  - Some URLs map to executable code that generates HTML
  - Program exits when HTML has been generated
  - Stateless servers:
    - Requests are independent
    - No state preserved between executions
  - Perl was a popular language for CGI applications
    - General purpose interpreted language, based on C

1st Generation Web App Frameworks

- Languages:
  - PHP
  - ASP (Microsoft - Application Server Pages)
  - Java Servlets
- Language runtime embedded in Web Server
- Frequently used templates to mix code and HTML / CSS
- Needed web-specific language extensions / packages
  - URL Handling
  - HTML generation
  - Stateful Sessions
  - Database Interaction
  - Etc.

2nd Generation Web App Frameworks

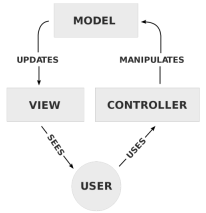
- Examples:
  - Ruby on Rails
  - Django
- Page generation still in Web Server
- Encouraged Model View Controller decomposition
  - We'll look at this later as it's still heavily used
- Typically supported object-relational mapping (ORM)
  - Simplifies database integration by exposing database contents (tables and rows) as classes and objects
  - Made generating dynamic pages easier

3rd Generation Web App Frameworks

- Example:
  - AngularJS
- JavaScript frameworks running in browser
  - No server-side page generation
- Frameworks now independent of server-side capabilities
  - Can write once and run in any (or at least, many) servers
- Support good ideas / concepts from earlier generations
  - Model View Controller
  - Templating
  - Separation of Style and Content

Model View Controller (MVC)

- A mature software design pattern for user interfaces
  - Originated in the 1970's
- **Model**
  - Takes care of the application's data
    - JavaScript objects
- **View**
  - Defines what the web page looks like
    - HTML & CSS
- **Controller**
  - Manipulates the model which causes changes in the view
    - JavaScript code



https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

3rd Generation Views

- However generated, browsers only render CSS styled HTML
  - This the fundamental principle of Web Applications
- Templates are a common page generation technique
  - Write HTML document with the parts of the page that are the same regardless of situation
  - Add tags indicating where sections of the page need to be changed for specific situations
  - The templating system replaces the tags with information relevant to the current situation
- Benefits of templates (compared to modifying DOM via JavaScript)
  - Easy to visualise document structure
    - Tagged HTML will still render in browser
  - Templating can occur in the server or in the browser

Template Example - Angular

- Angular has a rich templating syntax
  - Loops, conditionals, subroutines, …
- Simple example:

```
<body>
  <div class="greetings">
    Welcome back {{models.user.fname}}!,
  </div>
  <div class="applicationcount">
    You have {{models.application.count}}
      applications to approve.
  </div>
</body>
```


3rd Generation Controllers

- JavaScript running in browser
- Connect models and views
  - Communicate with server to get models and send updates
- Organise view templates
  - Control which ones are being shown
- Handle user interactions
  - Button clicks, menu activations, text entry, etc.

Controller Example - Angular

```
function userLandingView ($scope, $modelService) {
  $scope.models = {};
  $scope.models.users = $modelService.fetch("users");
  $scope.models.applications = $modelService.fetch("applications");
  $scope.okPushed = function okPushed() {
    // Code to execute when OK button is pushed
  }
}
```

`$scope` is the link between Angular Controller and View

https://docs.angularjs.org/guide/scope

3rd Generation Models

- All dynamic (i.e. non-static) information needed by the view templates or controllers
- Traditionally tied to application's database schema
  - In ORM a model is a row in a table
- Web application's model data needs are specified by the view designers but need to be persisted by the database
- Problem:
  - Traditional relational database schemas don't like changing
  - Web Application model data frequently changes
    - e.g. *"user will obviously like this view better if we add a A and get rid of B"*

Model Example - Angular

- Angular provides support for fetching data from the server
  - REST APIs (we'll see these later)
  - JSON is a common on-wire data representation
- Server representation is "whatever you like"
  - Angular is agnostic - it only cares about the on-wire format
  - e.g. Mongoose Object Definition Language (ODL) for MongoDB:

```
var userSchema = new Schema({
  firstName: String,
  lastName: String,
});
var User = mongoose.model('User', userSchema);
```


4th Generation Web App Frameworks

- Examples:
  - **React**, Vue, Angular 2
- Adopted techniques and concepts from earlier generations:
  - Browser side JavaScript
  - Model View Controller
  - Templating
- Concentrate on JavaScript components rather than HTML
- Write into a server-side Virtual DOM
  - When all modifications complete DOM replaced with contents of VDOM in a single operation ☺
    - Compare to tens of thousands of individual DOM modifications ☹

Assignment 2 - Review

- The browser is in charge
  - Ask for anything you like, but in the end the browser decides what you get ☹
- CSS is like any other programming language
  - You can write sloppy CSS, or you can write tight CSS
  - CSS has re-use capabilities

```
.letter {
  width: 60px;
  height: 60px;
  text-align: center;
  background-color: yellow;
  color: black;
}

.number {
  width: 60px;
  height: 60px;
  text-align: center;
  background-color: yellow;
  color: black;
}
```

☹

```
.letter, .number {
  width: 60px;
  height: 60px;
  text-align: center;
  background-color: yellow;
  color: black;
}
```

☺

Assignment 3 - Review

- Don't re-invent the wheel
  - Use Regular Expressions rather than write the logic yourself
- Testing is good
  - But be careful, easy to worry about edge cases that were not required
- Code Coverage

```
if (something) {
   // whatever
}
```

the `if` introduces a branch, both of which need to be covered

- Conflicts between coverage and linting

```
for (const prop in map) {
  if (map.hasOwnProperty(prop)) {
   // use prop
  }
  if (map.hasOwnProperty(prop) && prop != '')) {
   // use prop
  }
```

Google rules say we need to check prop before using it

Whoops! Uncovered branch ⊗

Branch now covered ☺

```
test('Undefined Tag', () => {
  const t = new Templater('Mary {{had}} a {{little}} {{lamb}}');
  expect(t.apply({'': ''})).toBe('Mary a');
});
```


Assignment 4 - Introduction

- Use everything we've looked at so far ☺
  - HTML, CSS, JSON, JavaScript, DOM, Events (woo-hoo!)
- It's significantly more work than Assignment 3 ☹
  - But not impossible, so take your time and keep calm
  - Get started **TODAY!**
- Three Components:
  - Basic: 3 points
    - Fill in an HTML table from JSON
  - Advanced: 2 points (1 for functionality 1 for look)
    - Write a simple date Picker
  - Stretch 2 points (1 for functionality 1 for look)
    - Write a (slightly) more sophisticated date Spinner
- You may want to write your own tests for Advanced & Stretch

Assignment 4 - Basic

{{H1}}	{{H2}}	{{H3}}
{{R11}}	{{R12}}	{{R13}}
{{R21}}	{{R22}}	{{R23}}
{{R31}}	{{R32}}	{{R33}}
{{R41}}	{{R42}}	{{R43}}
{{R51}}	{{R52}}	{{R53}}

```
{
  "H1":"Name",
  "H2":"Address",
  "H3":"Age",
  "R31":"Billy",
  "R22":"14a Main st.",
  "R53":"32"
}
```

By Tag | By Id

Click Either Button

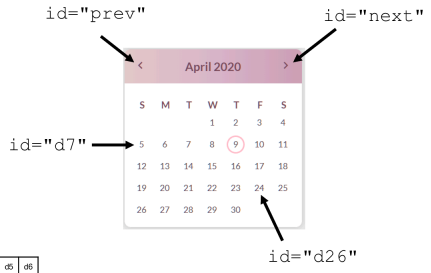
Name	Address	Age
Billy	14a Main st.	32

```
{
  "H1":"Name",
  "H2":"Address",
  "H3":"Age",
  "R31":"Billy",
  "R22":"14a Main st.",
  "R53":"32"
}
```

By Tag | By Id

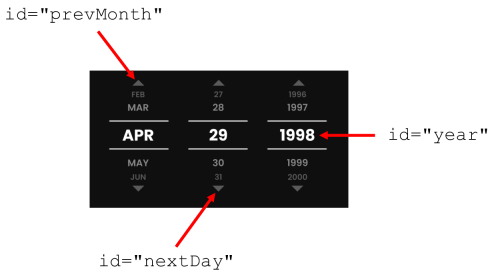
After first click, only "By Id" will do anything

Assignment 4 - Advanced - "Picker"



d0	d1	d2	d3	d4	d5	d6
d7	d8	d9	d10	d11	d12	d13
d14	d15	d16	d17	d18	d19	d20
d21	d22	d23	d24	d25	d26	d27
d28	d29	d30	d31	d32	d33	d34
d35	d36	d37	d38	d39	d40	d41

Assignment 4 - Stretch - “Spinner”

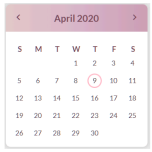


Assignment 4 - Advanced & Stretch

- Work on the functionality ***first***

<
October 2020
>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

- Then work on what they look like
  - Separate style from content ☺



Upcoming Lectures

- Monday: Introduction to React
- Wednesday: Single Page Web Applications
- Friday: Quiz 2

Tasks

- **Assignment 4** due 23:59 **Thursday October 29**

Web Applications

CSE183

Fall 2020

Introduction to React

Baskin Engineering
UC SANTA CRUZ

Notices

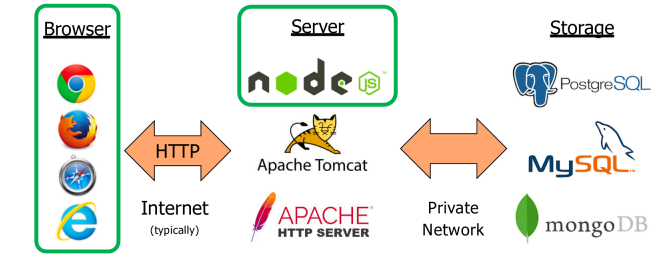
- **Assignment 4** due 23:59 **Thursday October 29**
- **Quiz 2** during class **Friday October 30**

Today's Lecture

- React Basics
- React Components & JavaScript XML (JSX)
- React Event Handling
- JSX Programming
- Component Lifecycle
- Stateless Components
- Component Communication

- Questions

Full Stack Web Applications



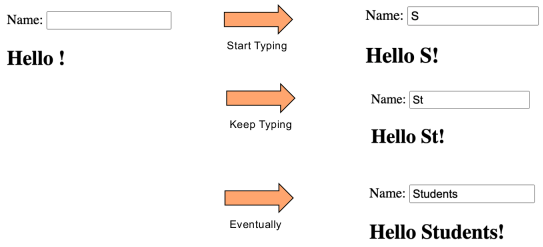
React Basics

- JavaScript framework for writing the web applications
  - Quick responses when running in browser
  - Less opinionated that other 4th Gen Web App Frameworks
    - Restricts itself to rendering views and handling user interactions
    - You can use any server and/or database you like ☺
- Uses the Model View Controller pattern
  - View constructed from Components
  - Supports HTML templating (optional, but heavily used)
- Minimal server-side support required
  - Most React apps can be deployed in any web server
- Intended for large and single page applications
  - Has modules, reusable components, testing hooks, etc.

Why "React"?

- Because it simplifies reacting to changes of state
- Side benefit:
  - Gives you the ability to keep application state away from the DOM
    - But *you* must code in a way that makes use of this

A Basic HTML / JavaScript App



A Basic HTML / JavaScript App

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>CSE183 HTML Basic</title>
  </head>
  <body>
    <script type="text/javascript" src="basic.js"></script>
  </body>
</html>
```

```
const label = document.createElement('label');
const input = document.createElement('input');
const message = document.createElement('h2');

label.textContent = 'Name: ';
input.setAttribute('type', "text");
message.textContent = 'Hello !';

document.body.appendChild(label);
document.body.appendChild(input);
document.body.appendChild(message);

function handleInput(event) {
  message.textContent = `Hello ${event.target.value}!`;
}

input.addEventListener('input', handleInput);
```

Name:

Hello !

A Basic React App

We Write:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>CSE183 React Basic</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

React Generates:

But How?

¯_(ツ)_/¯

Basic React App

Folders & Files

```
build  ( generated app ends up here )
public
  index.html
src
  index.js
  App.js
package.json
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>CSE183 React Basic</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

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


Node.js Config: `package.json`

```
{
  "name": "cse183-react-basic",
  "version": "1.0.0",
  "description": "CSE183 React Basic",
  "author": "David Harrison <dcharris@ucsc.edu>",
  "license": "UNLICENSED",
  "repository": "none",
  "devDependencies": {
    "react-scripts": "^4.0.0"
  },
  "dependencies": {
    "react": "^17.0.1",
    "react-dom": "^17.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Meta-data for this JavaScript package

3rd party development tools

3rd party packages needed at runtime

Possible arguments to 'npm run'

Browsers we'd like the production version to work in

Browsers we'd like the development version to work in

React Components

src/index.js

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

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

src/**App.js**

```
import React from "react";

class App extends React.Component {
  state = {
    name: null
  };
  constructor(props) {
    super(props);
  }
  render() {
    ...
  }
}

export default App;
```

React Component Rendering

With Intermediate Variables:

```
render() {
  const label = React.createElement('label', null, 'Name: ');

  const input = React.createElement('input',
    { type: 'text', value: this.state.name });

  const message = React.createElement('h1', null,
    'Hello ', this.state.name, '!');

  return React.createElement('div', null, label, input, message);
}
```

Whenever `this.state.name` changes, content of the `message` element ***automatically*** changes ☺☺☺☺☺

Thank you React!

React Component Rendering

Without Intermediate Variables:

```
render() {
  return React.createElement('div', null,
    React.createElement('label', null,'Name: '),
    React.createElement('input',
      { type: 'text', value: this.state.name }),
    React.createElement('h1', null,
      'Hello ', this.state.name, '!');
    );
}
```


JavaScript XML (JSX)

- A syntax extension to JavaScript
- Recommend mechanism for describing UIs in React
- Akin to template languages
  - But it comes with the full power of JavaScript ☺
- *"This funny tag syntax is neither a string nor HTML"*

```
const element = <h1>Hello, world!</h1>;
```


https://reactjs.org/docs/jsx-in-depth.html

React Component Rendering

With JSX:

```
render() {
  return (
    <div>
      <label>JSX Name: </label>
      <input
        type="text"
        value={this.state.name}
      />
      <h1>Hello {this.state.name}!</h1>
    </div>
  );
}
```

React Event Handling Problem

```
class App extends React.Component {
  ...
  handleInput(event) {
    this.setState({ name: event.target.value });
  }
  ...
}
```

Consider:

```
<input type="text" onChange={this.handleChange}/>
```

Problem! `this` has no scope ☹

React Event Handling

Arrow functions to the rescue ☺

```
class App extends React.Component {
  ...
  handleInput = (event) => {
    this.setState({ name: event.target.value });
  }
  render() {
    return (
      <div>
        <label>JSX Name: </label>
        <input
          type="text"
          value={this.state.name}
          onInput={(event) => this.handleInput(event)}
        />
        <h1>Hello {this.state.name}!</h1>
      </div>
    );
  }
}
```


React & JSX Code Conventions

- Choice of word separator in multi-word variable name:
  - `camelCase` vs. `dash-case`
- JavaScript is case sensitive, but HTML is not
- JSX looks like HTML but is more like JavaScript
- Recommend using `camelCase` for attributes in JSX
  - E.g. `return <CustomButton color="fireBrick" />;`

JSX Programming Fundamentals

- JSX is parsed into `React.createElement()` calls
  - Very important concept, try to remember it
- `React.createElement(type, props, ...children);`
  - `type`: HTML tag (e.g. `h1`, `p`) or any `React.Component` sub-class
  - `props`: attributes (e.g. `type="text"`) (must be in `camelCase`)
  - `children`: Zero or more children which can be :
    - Strings
    - Numbers
    - React Components
    - An array of the above

JSX Template Restrictions

- JSX Templates ***must*** return a valid `children` parameter
- Templates can have JavaScript scope variables and expressions

```
<div>{foo}</div>
```

  - Valid if `foo` is in scope (i.e would be a valid `createElement() childen` parameter)

```
<div>{foo + ' and ' + bar()}</div>
```

  - Valid if `foo` & `bar()` are in scope

- Template must evaluate to a value

```
<div>{if (something) { ... } }</div>
```

  - Doesn't work because `if` has no value
  - Same problem with `for` loops
    - And any other JavaScript statement that does not return a value
- Leads to contorted JSX ☹
  - E.g. Anonymous immediate functions

```
<div>{ (function() { if ...; for ...; return val;})() }</div>
```


JSX Conditional Rendering

- Use JavaScript Ternary operator (`?:`)

```
<div>{this.state.french ? <b>Bonjour</b> : "Hello"}</div>
```

- Use JavaScript variables

```
let greeting;
const en = "Hello";
const fr = <b>Bonjour</b>;
let {french} = this.state;
if (french) {
    greeting = fr
} else {
    greeting = en
};
<div>{greeting}</div>
```

Much Better ☺

JSX Iteration

- Use JavaScript array variables

```
let lis = [];
for (let i = 0; i < data.length; i++) {
  lis.push(<li key={data[i]}>Data Value {data[i]}</li>);
}
return <ul>{listItems}</ul>;
```

- Functional programming

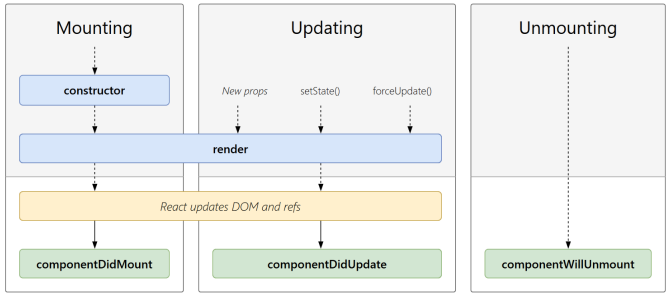
```
<ul>{data.map((d) => <li key={d}>Data Value {d}</li>)}</ul>
```


JSX Styling

```
import React from 'react';
import './MyComponent.css';

class MyComponent extends React.Component {
  ...
  render() {
    return (
      <span className="whatever">
      ...
     </span>
   );
  }
  ...
}
```

React Component Lifecycle



https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

React Component Lifecycle

```
class MyComponent extends React.Component {
  state {
    counter: 0
  }
  ...
  componentDidMount() { // Start 2 sec timer to increment a counter
    const incFunc = () => this.setState(
      { counter: this.state.counter + 1 });
    this.timerID = setInterval(incFunc, 2 * 1000);
  }
  componentWillUnmount() { // Cancel the timer
    clearInterval(this.timerID);
  }
  ...
}
```


React Stateless Components

- Components with properties but no methods can be functions rather than classes:

```
function MyComponent(props) {
  return <div>My name is {props.name}</div>;
}
```

- Or destructured:

```
function MyComponent({name}) {
  return <div>My name is {name}</div>;
}
```

- React Hooks
  - Add state to stateless components
  - https://reactjs.org/docs/hooks-intro.html

React Component Communication

- Parent to child:
  - Properties (attributes)

```
<ChildComponent param={dataForChild} />
```

- Child to parent:
  - Callbacks

```
this.parentCallback = (dataFromChild) => {
    /* process dataFromChild */
};
<ChildComponent callback={this.parentCallback}> />
```

- React Context
  - Global variables for subtree of components
  - https://reactjs.org/docs/context.html

Upcoming Lectures

- Wednesday: Single Page and Responsive Web Applications
- Friday: Quiz 2 & Assignment 5 Introduction

Tasks

- **Assignment 4** due 23:59 **Thursday October 29**
- Study for **Quiz 2** during class **Friday October 30**

CSE183

Lecture will start at 09:20

Baskin Engineering UC SANTA CRUZ

Notices

- **Assignment 4** due 23:59 **Thursday October 29**
- **Quiz 2** during class **Friday October 30**

Web Applications

CSE183

Fall 2020

Single Page Applications

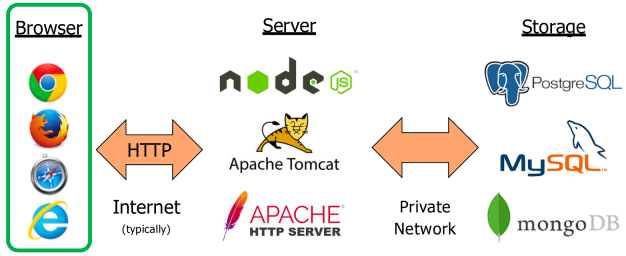
Baskin Engineering UC SANTA CRUZ

Today's Lecture

- React Review
- Problems with Multipage Web Apps
- Deep Linking
- Single Page Applications (SPA)
- React Routers and SPA

- Assignment 4 - Secret Sauce 
- Questions

Full Stack Web Applications



React Review

- JavaScript framework for web applications
- Uses the Model View Controller pattern
- Minimal server-side support required
- Intended for large and **single page applications**
- Simplifies reacting to changes of state
- Includes JSX Templating
- Handles Component Lifecycles

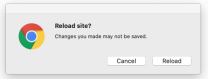
Problems with Multipage Web Apps

- By definition, web applications run in browsers
- Users are familiar with browsers ☺
  - A history of URLs visited is maintained
    - Back & Forward buttons - navigate the URL history
  - Easily move to a new page
    - Type into location bar or use forward/back buttons
    - Selecting a bookmarked URL
    - Clicking the page refresh button
- Browser deletes the current JavaScript context when user moves to a different page
  - Necessary to preserve resources and for security, but…
  - Problematic for JavaScript frameworks ☹
    - URL (and cookies) are the only information preserved

Problems with Multipage Web Apps

- Static Apps: HTML Pages served from web server
  - Each page had a URL and app switched between
- Early Dynamic Apps: JavaScript pages
  - Problem: Restart web app on navigation (could loose all your work ☹)

```
window.addEventListener('beforeunload', function (e) {
  e.preventDefault();
  e.returnValue = '';
});
```



- Users expect app to support well-known behaviors:
  - Navigate with forward & back buttons and page history
  - Navigate away and come back to the app
  - Bookmark a specific place in the app
  - Copy the URL from the location bar and share it
  - Click the page refresh button and application state is maintained

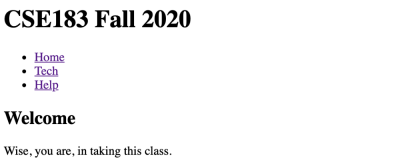
MPA in HTML

Folders & Files

```
help.html
tech.html
index.html
common.css
```


index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML / JavaScript MPA</title>
  <link href="common.css" rel="stylesheet"/>
</head>
<body>
  <h1>CSE183 Fall 2020</h1>
  <ul class="navigation">
    <li><a href="index.html" class="active">Home</a></li>
    <li><a href="tech.html">Tech</a></li>
    <li><a href="help.html">Help</a></li>
  </ul>
  <h2>Welcome</h2>
  <p>Wise, you are, in taking this class.</p>
</body>
</html>
```



CSE183 Fall 2020

- Home
- Tech
- Help

Welcome

Wise, you are, in taking this class.

help.html

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML / JavaScript MPA</title>
  <link href="common.css" rel="stylesheet"/>
</head>
<body>
  <h1>CSE183 Fall 2020</h1>
  <ul class="navigation">
    <li><a href="index.html">Home</a></li>
    <li><a href="tech.html">Tech</a></li>
    <li><a href="help.html" class="active">Help</a></li>
  </ul>
  <h2>Got Questions?</h2>
  <p>Easiest thing to do is mail me a letter.</p>
</body>
</html>
```

CSE183 Fall 2020

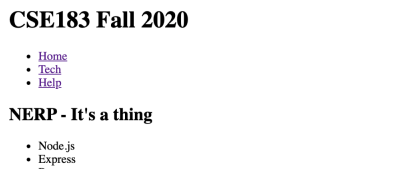
- Home
- Tech
- Help

Got Questions?

Easiest thing to do is mail me a letter.

tech.html

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML / JavaScript MPA</title>
  <link href="common.css" rel="stylesheet"/>
</head>
<body>
  <h1>CSE183 Fall 2020</h1>
  <ul class="navigation">
    <li><a href="index.html">Home</a></li>
    <li><a href="tech.html" class="active">Tech</a></li>
    <li><a href="help.html">Help</a></li>
  </ul>
  <h2>NERP - It's a thing</h2>
  <ul>
    <li>Node.js</li>
    <li>Express</li>
    <li>React</li>
    <li>PostgreSQL</li>
  </ul>
</body>
</html>
```



CSE183 Fall 2020

- Home
- Tech
- Help

NERP - It's a thing

- Node.js
- Express
- React
- PostgreSQL

Change URL Without Reloading Page

- URL hash fragment can change without without reload

```
http://example.com
http://example.com#fragment
http://example.com?id=1234
http://example.com?id=1234#fragment
```

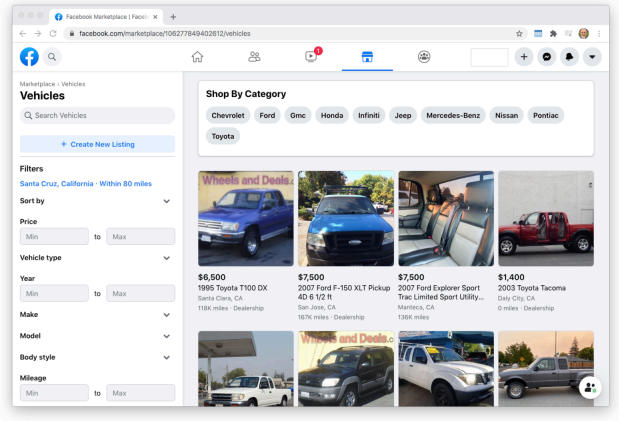
- HTML5 Browser History API via `window.history`
  - Gives JavaScript control of page reload
  - Allows change of URL without reloading page

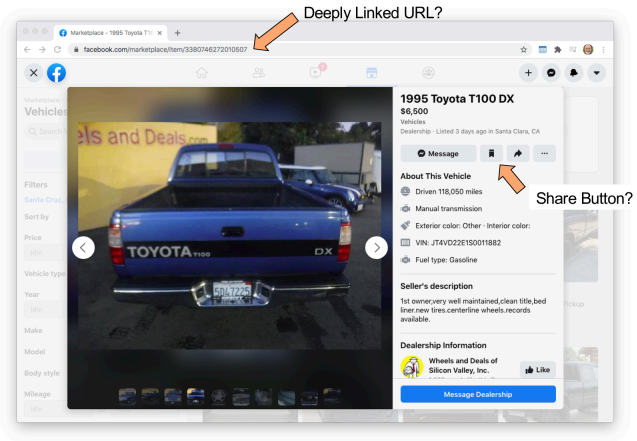
Deep Linking

- General Idea:
  - URLs should capture a web app's context so that browsing to a particular URL persuades the app to switch to the relevant context
  - Supports:
    - Bookmarks (private (ish))
    - Sharing (public)
- App context is defined by the user interface designer:
  - E.g. Viewing a data entity with an edit dialog open
    - Should the link point to the data entity view or to the entity & dialog?
    - Does it matter if user bookmarking for personal use or sharing with others?
    - How about navigating away and back or browser refresh?

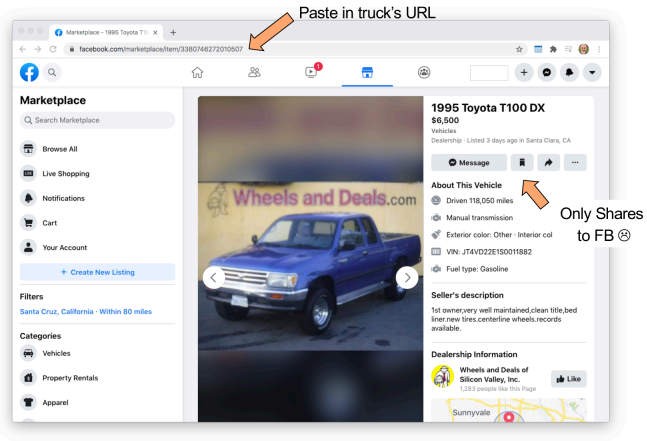
Deep Linking in Single Page Apps

- Encapsulate the app's context in the URL
  - Works for browser navigation and refresh
  - User can copy URL from location bar
- Provide a share button to generate deeply linked URL
  - Allows user to explicitly fetch a URL based on need
  - Can keep URL in location bar pretty
- In both cases the web app should be able to initialize itself from a deeply linked URL

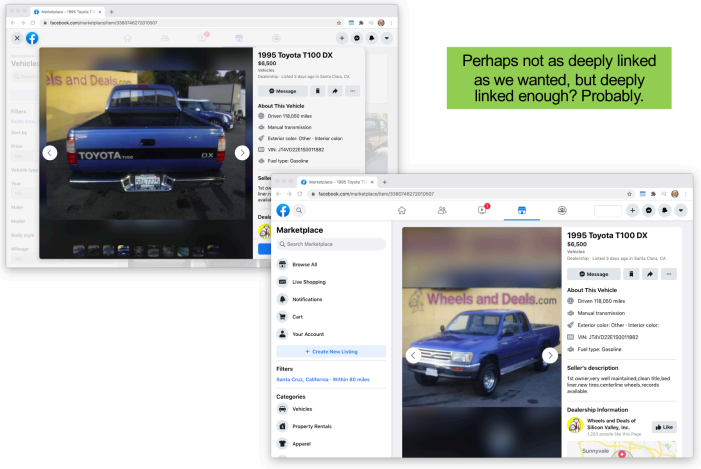




18



19



Perhaps not as deeply linked as we wanted, but deeply linked enough? Probably.

20

URL Ugliness

- Compare:
  `http://www.example.org/dirmod?sid=789AB8&type=gen&mod=Core+Pages&gid=A6CD4967199`
- To:
  `http://www.example.org/show/A6CD4967199`
- And what the hey is this?
  `https://www.flickr.com/photos/dcharris/35780000165/in/explore-2020-10-228/`

21

SPA in HTML / JavaScript

Folders & Files

```
Help.js
Home.js
Tech.js
index.html
index.css
index.js
```


Home.js

```
class Home {
  constructor(containerId) {
    const container = document.getElementById(containerId);
    const h2 = document.createElement('h2');
    h2.textContent = 'Welcome';
    const p = document.createElement('p');
    p.textContent = 'Wise, you are, in taking this class.';
    container.appendChild(h2);
    container.appendChild(p);
  }
}
```

Welcome

Wise, you are, in taking this class.

Help.js

```
class Help {
  constructor(containerId) {
    const container = document.getElementById(containerId);
    const h2 = document.createElement('h2');
    h2.textContent = 'Got Questions?';
    const p = document.createElement('p');
    p.textContent = 'Easiest thing to do is mail me a letter.';
    container.appendChild(h2);
    container.appendChild(p);
  }
}
```

Got Questions?

Easiest thing to do is mail me a letter.

Tech.js

```
class Tech {
  constructor(containerId) {
    const container = document.getElementById(containerId);
    const h2 = document.createElement('h2');
    h2.textContent = `NERP - It's a thing`;
    const ul = document.createElement('ul');
    for (const item of ['Node.js','Express','React','PostgreSQL']) {
      const li = document.createElement('li');
      li.textContent = item;
      ul.appendChild(li);
    }
    container.appendChild(h2);
    container.appendChild(ul);
  }
}
```

NERP - It's a thing

- Node.js
- Express
- React
- PostgreSQL

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML / JavaScript SPA</title>
  <link href="index.css" rel="stylesheet"/>
  <script type="text/javascript" src="index.js"></script>
  <script type="text/javascript" src="Home.js"></script>
  <script type="text/javascript" src="Tech.js"></script>
  <script type="text/javascript" src="Help.js"></script>
</head>
<body>
  <h1>CSE183 Fall 2020</h1>
  <ul class="navigation">
    <li><a id="homeRef" href="#/home">Home</a></li>
    <li><a id="techRef" href="#/tech">Tech</a></li>
    <li><a id="helpRef" href="#/help">Help</a></li>
  </ul>
  <div>
    <div id="home"></div>
    <div id="tech"></div>
    <div id="help"></div>
  </div>
</body>
<script>
  new Home("home");
  new Tech("tech");
  new Help("help");
</script>
</html>
```

CSE183 Fall 2020

- Home
- Tech
- Help

Welcome

Wise, you are, in taking this class.

index.js

```
function hide(...ids) {
  for (const id of ids) {
    document.getElementById(id).style.display = 'none';
    document.getElementById(id+'Ref').className = '';
  }
}

function hideAll() {
  hide('home','tech','help');
}

function show(id) {
  hideAll();
  document.getElementById(id).style.display = 'inline';
  document.getElementById(id+'Ref').className = 'active';
}

function attachListeners() {
  document.getElementById('homeRef').addEventListener('click', function (e) { show('home'); });
  document.getElementById('techRef').addEventListener('click', function (e) { show('tech'); });
  document.getElementById('helpRef').addEventListener('click', function (e) { show('help'); });
}

window.addEventListener('DOMContentLoaded', function (e) {
  attachListeners();
  show(window.location.hash ? window.location.hash.slice(2) : 'home');
});
```


React Support for SPA

- React remains firmly unopinionated
  - If you want SPA, do it yourself or use a 3rd party library
- React Router https://reactrouter.com
  - Uses URL to control conditional rendering
    - `HashRouter` : Fragment part of the URL
    - `BrowserRouter` : HTML5 URL Handler

React Router Example

Folders & Files

```
build
public
   index.html
src
   component
      Help.js
      Home.js
      Tech.js
   App.js
   index.js
package.json
```

New

src/component/Home.js

```
class Home extends React.Component {
  render() {
    return (
      <div>
        <h2>Welcome</h2>
        <p>Wise, you are, in taking this class.</p>
      </div>
    );
  }
}
```

Welcome

Wise, you are, in taking this class.

src/component/Help.js

```
class Home extends React.Component {
  render() {
    return (
      <div>
        <h2>Got Questions?</h2>
        <p>Easiest thing to do is mail me a letter.</p>
      </div>
    );
  }
}
```

Got Questions?

Easiest thing to do is mail me a letter.

src/component/Tech.js

```
class Tech extends React.Component {
  render() {
    return (
      <div>
        <h2>NERP - It's a thing</h2>
        <ul>
          <li>Node.js</li>
          <li>Express</li>
          <li>React</li>
          <li>PostgreSQL</li>
        </ul>
      </div>
    );
  }
}
```

NERP - It's a thing

- Node.js
- Express
- React
- PostgreSQL

src/App.js

```
import React from "react";
import {Route, NavLink, HashRouter} from "react-router-dom";

import Home from "./component/Home";
import Tech from "./component/Tech";
import Help from "./component/Help";

class App extends React.Component {
  render() {
    return (
      <HashRouter>
        <h1>CSE183 Fall 2020</h1>
        <ul className="navigation">
          <li><NavLink exact to="/">Home</NavLink></li>
          <li><NavLink to="/tech">Tech</NavLink></li>
          <li><NavLink to="/help">Help</NavLink></li>
        </ul>
        <div>
          <Route exact path="/" component={Home}/>
          <Route path="/tech" component={Tech}/>
          <Route path="/help" component={Help}/>
        </div>
      </HashRouter>
    );
  }
}
```

CSE183 Fall 2020

- Home
- Tech
- Help

Welcome

Wise, you are, in taking this class.

React Router Example

CSE183 Fall 2020

Home | Tech | Help

NERP - It's a thing

- Node.js
- Express
- React
- PostgreSQL

Code for all three examples available for download after lecture

Assignment 4 - Secret Sauce

```
<head>
  <title> CSE183 Assignment 4 - Basic </title>
  <link href="templater.css" rel="stylesheet" />
  <script type="text/javascript" src="Templater.js"></script>
</head>
<body>
  <table>
    ...
  </table>
  <div id="hidden">{{trl1}}</div>
  <textarea id="json" rows="8", cols="30">{ ... }</textarea>
  <button id="byTag">By Tag</button>
  <button id="byId">By Id</button>
</body>
</html>
```

Need to create a `Templater` instance and connect the By Tag By Id buttons to it

Assignment 4 - Secret Sauce

```
<head>
  <title> CSE183 Assignment 4 - Basic </title>
  <link href="templater.css" rel="stylesheet" />
  <script type="text/javascript" src="Templater.js"></script>
</head>
<body>
  <table>
    ...
  </table>
  <div id="hidden">{{tr11}}</div>
  <textarea id="json" rows="8", cols="30">{ ... }</textarea>
  <button id="byTag">By Tag</button>
  <button id="byId">By Id</button>
</body>
<script>
  const templater = new Templater();
  document.getElementById("byTag").addEventListener('click', function(e) {
    templater.byTag(document, document.getElementById("json").value);
  });
  document.getElementById("byId").addEventListener('click', function(e) {
    templater.byId(document, document.getElementById("json").value);
  });
</script>
</html>
```

Problem?

The By Tag By Id buttons exist before the event handlers are attached ☹

Assignment 4 - Secret Sauce

```
<head>
  <title> CSE183 Assignment 4 - Basic </title>
  <link href="templater.css" rel="stylesheet" />
  <script type="text/javascript" src="Templater.js"></script>
  <script>
    const templater = new Templater();
    document.getElementById("byTag").addEventListener('click', function(e) {
      templater.byTag(document, document.getElementById("json").value);
    });
    document.getElementById("byId").addEventListener('click', function(e) {
      templater.byId(document, document.getElementById("json").value);
    });
  </script>
</head>
<body>
  <table>
    ...
  </table>
  <div id="hidden">{{tr11}}</div>
  <textarea id="json" rows="8", cols="30">{ ... }</textarea>
  <button id="byTag">By Tag</button>
  <button id="byId">By Id</button>
</body>
</html>
```

Problem?

The By Tag By Id buttons do not exist when we try to attach the event handlers ☹

Assignment 4 - Secret Sauce

```
<head>
  <title> CSE183 Assignment 4 - Basic </title>
  <link href="templater.css" rel="stylesheet" />
  <script type="text/javascript" src="Templater.js"></script>
  <script>
    const templater = new Templater();
    function byTag() {
      templater.byTag(document, document.getElementById("json").value);
    }
    function byId() {
      templater.byId(document, document.getElementById("json").value);
    }
  </script>
</head>
<body>
  <table>
    ...
  </table>
  <div id="hidden">{{tr11}}</div>
  <textarea id="json" rows="8", cols="30">{ ... }</textarea>
  <button id="byTag" onclick="byTag()">By Tag</button>
  <button id="byId" onclick="byId()">By Id</button>
</body>
</html>
```

Problem Solved

Attach event handlers as the buttons are being created ☺

Assignment 4 - Secret Sauce

```
<head>
  <title> CSE183 Assignment 4 - Basic </title>
  <link href="templater.css" rel="stylesheet" />
  <script type="text/javascript" src="Templater.js"></script>
 <script>
  window.addEventListener('DOMContentLoaded', function (e) {
    const templater = new Templater();
    document.getElementById('byTag').addEventListener('click', function (e) {
      templater.byTag(document, document.getElementById("json").value);
    });
    document.getElementById('byId').addEventListener('click', function (e) {
      templater.byId(document, document.getElementById("json").value);
    });
  });
  </script>
</head>
<body>
  <table>
    ...
  </table>
  <div id="hidden">{{tr11}}</div>
  <textarea id="json" rows="8", cols="30">{ ... }</textarea>
  <button id="byTag">By Tag</button>
  <button id="byId">By Id</button>
</body>
</html>
```

Alternate Solution

Attach event handlers after DOM is loaded ☺

Upcoming Lectures

- Friday: Quiz 2
  - Plus Assignment 4 Review & Assignment 5 Introduction
- Monday: React Global State & Responsive Web Applications

Tasks

- **Assignment 4** due 23:59 **Thursday October 29**
- Study for **Quiz 2** during class **Friday October 30**