Seneca College

Web Programming

WEB322 – Week 11 – Class 1

WEB322 Hc

Introduction to jQuery & Bootstrap Frameworks

From the jQuery website:

> "jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people wri JavaScript."

Essentially, jQuery is an extremely popular JS library that we can use to simplify many of the client-side tasks that we learned i WEB222 for **DOM manipulation**, **Event Handling** and **AJAX**. It also has a large community of developers working on free/oper "plugins" (reusable jQuery code) that makes incorporating complex UI components & UX functionality simple. For example: https://fullcalendar.io provides all of the source code necessary to create a fully featured drag/drop calendar component. This and a ton more can be found on a central "Plugin Registry" (many available on NPM) online.

Including jQuery in our Projects

To incorporate the jQuery library into our assignments, we simply need to add a reference to it from our HTML page (just like other "external" JavaScript file / files). This can be accomplished either by downloading the source files and including them in c solution or using a CDN (Content Delivery Network). For example, if we decided to download the minified version of the latest to include in our projects (usually in /js/lib/jQuery), we would use the code:

```
<head>
    <script src="/js/lib/jQuery/jquery-3.2.1.min.js"></script>
    <!-- ... -->
</head>
```

If we wanted to use the same version from the CDN, we would use the following code instead:

```
<head>
    <script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4gsxgFZhOsE
    <!-- ... -->
</head>
```

Client Side JS & $(document).ready()

Now that we have the jQuery library correctly added to our view, we should add another external JS file that uses the library. In case, we will add a **"main.js"** file directly under the **"js"** directory (since it isn't a library / plugin) and include it **beneath** jQuery. very important, bccause if we accidentally place code that *uses* jQuery before the code to include the jQuery library, we will ge Therefore, we must include any external JS files that will be using the jQuery library **beneath** our jQuery <script> element:

```
<head>
    <script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4gsxgFZhOsE
    <script src="/js/main.js"></script>
    <!-- ... -->
</head>
```

To test to make sure everything is working properly, we will write an anonymous *callback* function (inside main.js) and provide **$(document).ready()** method:

```
$(document).ready(function(){
    console.log("document ready!");
});

// alternatively:

// $(function() {
//    console.log( "document ready!" );
// });

console.log("file loaded");
```

If we try running our file in the web browser with the console open, we should see the messages: "file loaded" followed by "do ready!". This is because the (callback) function provided to the **$(document).ready** function contains text that will *only* output the **document is ready**, ie: **when the DOM is ready and safe to manipulate**. Since we will be primarily be using the DOM (upc nodes, wiring up events, etc), we must ensure that all of our jQuery code is written *inside* a callback provided to $document.re From the documentation:

> "A page can't be manipulated safely until the document is "ready." jQuery detects this state of readiness for you. Code included inside $(document).ready() will only run once the page Document Object Model (DOM) is ready for JavaScript code to execute. Code included inside $(window).on("load", function() { ... }) will run once the entire page (images or iframes), not just the DOM, is ready."

This is why you will see most jQuery examples written in the pattern:

```
$(function() {
```

```
    // do something cool...
});
```

The **dollar sign ($)** syntax is just a shortcut for **jQuery**. So the above code could be re-written as **jQuery(function(){ ... });**, howe is not as common. Typically, the dollar sign ($) syntax is left intact, unless it is conflicting with another client-side JS library (pro MooTools, YUI, etc.), in which case, the .noConflict() function is used and we abandon the $.

jQuery Selectors

One of the most valuable features provided by jQuery is it's comprehensive and powerful **selectors** which provide a fast way c accessing elements in the DOM using CSS-style syntax, similar to **document.querySelector()** and **document.querySelectorAll** However, jQuery selectors have been expanded to include more flexibility and cross-browser compatibility. Additionally, since selectors return one or more objects that ***wrap*** **native DOM elements**, we also gain access to a number of functions to easily with the result set (ie: watching for events / modifying the DOM).

Some common selectors that jQuery gives us are:

Selector	Description
$("*")	**All Selector:** Selects all elements
$("#myDiv")	**id Selector:** Selects a single element with the given id attribute.
$(".myClass")	**class Selector** Selects all elements with the given class.
$(":input")	**input Selector** Selects all input, textarea, select and button elements.
$(":radio")	**radio Selector** Selects all elements of type radio.
$(":checkbox")	**checkbox Selector** Selects all elements of type checkbox.
$(":visible")	**visible Selector** Selects all elements that are visible.
$(":hidden")	**hidden Selector** Selects all elements that are hidden (the opposite of :visible).
$(":odd") ie: $("tr:odd")	**odd Selector** Selects odd elements, zero-indexed. See also even.
$(":has(selector)") ie: $("div:has(p)")	**has() Selector** Selects elements which contain at least one element that matches th specified selector.
For a full list of the *60+* selector types, refer to: https://api.jquery.com/category/selectors	

Accessing the Selected Elements

As discussed above, using one (or more) of the above selectors gives us access to (jQuery wrapped) DOM elements. Using this either work with the **whole collection** of returned results, ie:

```
$(document).ready(function(){
    // make all <li> elements inside <ul class="list1">...</ul> bold
    $("ul.list1 li").css("font-weight", "bold");
});
```

```
});
```

Access **each element** individually using .each() & $(this):

```
$(document).ready(function(){
    // append each <li> element inside <ul class="list1">...</ul>
    // with it's position in the list
    $( "ul.list1 li" ).each(function( index ) { // DO NOT use () => {} syntax here
        $(this).append(" " + index);
    });
});
```

Filter the results down further using .filter():

```
$(document).ready(function(){
    // make all odd <li> elements inside <ul class="list1">...</ul> bold
    $("ul.list1 li").filter(":odd").css("font-weight", "bold");
});
```

Event Handling

An important part of web programming is the ability execute code when a certain "event" occurs (ie: a button is pressed, a for submitted, a value changed, the user swiped up, etc, etc.). The act of registering a (callback) function to a specific event is often "wiring" up the event, in the same way that we would wire up a light bulb to a light switch. Fortunately, jQuery provides a very way to add/remove logic from an event, as well as exposing a wide range of events to choose from:

- Keyboard Events
- Mouse Events
- Form Events
- Browser Events
- Mobile Events (swipe, tap, etc)

If we wish to respond to one of the events listed above, we invoke the .on() method on the specific **element(s)** that we wish to the event to. For example, say we wish to change the font colour of a list element when it's "clicked":

```
$(document).ready(function () {
    $("ul.list1").on("click", "li", function () { // DO NOT use () => {} syntax here
        $(this).css("color", "red");
    });

    $("ul.list1").append("<li>I get the event too!</li>");
});
```

Notice how we can specify the event on a parent element (<ul class="list1">...) and provide a **selector** to specify the targe element(s) for the event? This syntax is important, because if we dynamically add an element to the list it will automatically ge

event as well! For example, say we wish to build DOM nodes dynamically that must respond to an event, such as table rows bu JSON data that show a tooltip when clicked? To ensure that every new row gets the click event, we specify the event on the tab provide a selector to handle the dynamically-added <tr> elements.

On the other hand, if we want to *remove* an event from an element, we simply invoke the .off() method on the element:

```
$(document).ready(function () {
    $("ul.list1").off("click", "li");
});
```

DOM Modification

Now that we know how to select elements from the DOM and wire events, it is important to discuss how we can actually **upda** DOM. We have seen this in the examples above using the .css() and .append(), however jQuery provides a host of other metho modify the DOM, including:

Property / Method	Description
.css()	Get the value of a computed style property for the first element in the set of matche elements or set one or more CSS properties for every matched element.
.append()	Insert content, specified by the parameter, to the end of each element in the set of matched elements.
.remove()	Remove the set of matched elements from the DOM.
.clone()	Create a deep copy of the set of matched elements.
.attr()	Get the value of an attribute for the first element in the set of matched elements or one or more attributes for every matched element.
.addClass()	Adds the specified class(es) to each element in the set of matched elements. Also se .removeClass()
.replaceWith()	Replace each element in the set of matched elements with the provided new conter return the set of elements that was removed.
.wrap()	Wrap an HTML structure around each element in the set of matched elements.
For a full list of the *40+* properties / methods used for DOM manipulation, refer to: http://api.jquery.com/category/manipul	

Using AJAX

AJAX is rapidly becoming one of the most important technologies in web programming. AJAX enables single-page applications more streamlined user interfaces for working with data. Recall (from week 9), creating an AJAX request with data and working response involved the following code (at minimum):

```
let httpRequest = new XMLHttpRequest();

httpRequest.open('POST', 'https://httpbin.org/post');
```

```
// set the Request Header and send the request
httpRequest.setRequestHeader("Content-Type", "application/json");
httpRequest.send(JSON.stringify({ user: "John Doe", job: "unknown" }));

httpRequest.onreadystatechange = function(){
    // check the current value of the readyState & status properties
    // if successful, process the data
    if (httpRequest.readyState === 4) { // The request has completed ("done")
        if (httpRequest.status === 200) {  // We successfully recieved the response
            // Create object jsData from the JSON received in responseText
            let jsData = JSON.parse(httpRequest.responseText);
            // render it in the console
            console.log(jsData);
        }else{
            console.log("error: " + httpRequest.statusText);
        }
    }
}
```

jQuery provides a much cleaner, cross-browser and backwards compatible approach with it's $.ajax() method:

```
$.ajax({
    url: "https://httpbin.org/post",
    type: "POST",
    data: JSON.stringify({ user: "John Doe", job: "unknown" }),
    contentType: "application/json"
})
.done(function (data) {
    console.log(data);
})
.fail(function (err) {
    console.log("error: " + err.statusText);
});
```

Bootstrap Framework

The Bootstrap framework is a set of **JavaScript** & **CSS** files that simplify the design of complex layouts & UI/UX functionality. It used as a starting point for modern websites, given its clean design patterns and unobtrusive JavaScript components. Bootstr has excellent documentation, making it simple for developers to prototype web apps quickly and efficiently. It is for these reas it's been so widely adopted by the industry as the de facto starting point when building everything from simple static sites to c web applications.

Including Bootstrap in our Projects

Like jQuery, to incorporate Bootstrap into our projects, we simply need to add some external files to our views and we can beg it right away. As with any external JavaScript or CSS files, we can choose to either download the files to our local project, or use (content delivery network).

It is important to note that Bootstrap **depends on jQuery** for it's interactive components, so if we wish to use anything beyond features of the Bootstrap framework, we must include jQuery as well:

Using a **local copy** - typically installed in "/lib/bootstrap":

```
<head>
    <link rel="stylesheet"  href="/lib/bootstrap/css/bootstrap.min.css">
    <!-- it is common to place the .js files at the end of the <body> tag as well -->
    <script src="/js/lib/jQuery/jquery-3.2.1.min.js"></script>
    <script src="/lib/bootstrap/js/bootstrap.min.js"></script>
    <!-- ... -->
</head>
```

Using a **CDN**:

```
<head>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap
    <!-- it is common to place the .js files at the end of the <body> tag as well -->
    <script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4gsxgFZhOsE
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrit
    <!-- ... -->
</head>
```

Responsive Grid System

Arguably one of the best features of the Bootstrap framework is it's Responsive Grid System. CSS Grid systems have risen in p in recent years because they allow designers to easily create visually pleasing, clean layouts without manually fiddling with flo margins, padding, flexbox, etc. Additionally, if a "responsive" grid system is used correctly, it can be very simple to create layou **also** conform to responsive design principles. Recall: responsive design can be defined as:

(img src: https://www.tutorialrepublic.com/twitter-bootstrap-tutorial/bootstrap-responsive-layout.php)

> "Responsive web design, originally defined by Ethan Marcotte in A List Apart, responds to the needs of the users and th devices they're using. The layout changes based on the size and capabilities of the device. For example, on a phone users would see content shown in a single column view; a tablet might show the same content in two columns."

Fortunately, Bootstrap's Grid System makes this task extremely simple, but still provides enough tools to create complex arrangements of elements based on viewport size.

To get started, we begin with a **.container** - this is the outermost block element that will contain all of the "rows" and "columns grid system as well as centre the content (grid) on the page:

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

Next, we must figure out how many "rows" we wish to include in our layout. For now, let's include two (2) rows:

```
<div class="container">
    <div class="row">
    </div>
    <div class="row">
    </div>
</div>
```

To complete our "grid" we must choose how many columns we would like to add (we can have a different number in each row Bootstrap, we can add a **maximum of twelve (12)** columns. If we wish to have fewer columns (ie: 3 columns), we tell each colu many of the 12 columns it should take up. For example, if we want to have three (3) columns, each column would be as wide a **(4) columns**, since 4 + 4 + 4 = 12. Similarly, if we only wanted to have two (2) columns, each column would be as wide as **six (6) columns**, since 6 + 6 = 12, and so on:

1	1	1	1	1	1	1	1	1	1	1	
8										4	
4				4						4	
6						6					

Once we have decided how many columns we want at the largest size, we must determine how each of those columns will **sca the viewport**. The most common configuration has the grid starting out stacked on mobile devices and tablet devices (the ext to small range) before becoming horizontal on desktop (medium and larger) devices.

To achieve this, we use the class **"col-md-*"** where ***** is how **wide** we want the columns to be at their (medium and larger) size. say that each of our rows will have three (3) columns - in the largest size, it would appear as:

.col-md-4	.col-md-4	.col-md-4

.col-md-4	.col-md-4	.col-md-4

However, in the mobile and tablet size (extra small to small range), our columns would appear stacked:

.col-md-4
.col-md-4
.col-md-4

.col-md-4
.col-md-4
.col-md-4

To implement this in our example from above, we simply add three (3) columns in each "row":

```
<div class="container">
    <div class="row">
        <div class="col-md-4"></div>
        <div class="col-md-4"></div>
        <div class="col-md-4"></div>
    </div>
    <div class="row">
        <div class="col-md-4"></div>
        <div class="col-md-4"></div>
        <div class="col-md-4"></div>
    </div>
</div>
```

Viewport Specific Configurations

If we want to be more specific with how the grids appear at each viewport size, we can use one or more of the following class on each row (* represents number of columns):

.col-xs-*	Extra small devices - Phones (< 768px)
.col-sm-*	Small devices - Tablets (≥ 768px)
.col-md-*	Medium devices - Desktops (≥ 992px)

.col-lg-*	Large devices - Desktops (≥ 1200px)

Offsetting Columns

Sometimes our design requires columns to be "offset" from the left of the grid. For example, if we wanted to only use the 4 mi columns, we would create a single "col-md-4" and offset it by four (4) columns from the left. This can be accomplished with Bo **.col-x-offset-y** classes, where **x is the target size** (ie, "sm", "md", etc.) and **y is the number of columns** (1 - 12). For example (fr Boostrap documentation):


```
<div class="container">
    <div class="row">
        <div class="col-md-4">.col-md-4</div>
        <div class="col-md-4 col-md-offset-4">.col-md-4 .col-md-offset-4</div>
    </div>
    <div class="row">
        <div class="col-md-3 col-md-offset-3">.col-md-3 .col-md-offset-3</div>
        <div class="col-md-3 col-md-offset-3">.col-md-3 .col-md-offset-3</div>
    </div>
    <div class="row">
        <div class="col-md-6 col-md-offset-3">.col-md-6 .col-md-offset-3</div>
    </div>
</div>
```

Note: As a final (but important) note about responsive design; Bootstrap also has created some **Responsive Utility Classes** th the visibility of elements to be toggled depending on each device size (ie: xs, sm, md, lg). Using these utilities in conjunction wi responsive grid system (as illustrated above), it is possible to implement a complex, responsive layout without writing any extr manage the configuration across device sizes!

Components

Bootstrap comes with a wide range of reusable components to help implement your design. They are all widely used, howeve only enough time to discuss the most interesting/important ones today:

Glyphicons

Bootstrap comes bundled with the premium icon font Glyphicons. Most modern web apps use icons to help the usability of th application, for example a "magnifying glass" (🔍) for searching, or a "floppy disk" (💾) to indicate saving. As a way to offer the as flexible a manner as possible (rendered "cleanly" at any size), special web fonts where introduced that contain the icons. Th where Glyphicons comes in - it is essentially a font that contains a large range of icons that we can use in our application. Sinc font (represented as a vector), we can size the icon up or down depending on our needs using the "font-size" property, withou

font (represented as a vector), we can size the icon up or down depending on our needs using the "font-size" property, without loss of quality:

(img src: http://glyphicons.com)

To incorporate an icon using Bootstrap's Glyphicons (often used in <button> elements), simply use the following code (in this c will use the "search" icon):

```
<span class="glyphicon glyphicon-search"></span>
```

Buttons

Another important "component" that Bootstrap provides is a set of classes to render buttons. There is no escaping the need fo buttons, whether they're hyperlinks (<a>...), buttons (<button>...</button>) or input type=submit / button buttons (<inp type="submit" />). Once again, Bootstrap comes to the rescue with a set of classes to create consistent, clean buttons:


```
<!-- Standard button -->
<button type="button" class="btn btn-default">Default</button>

<!-- Provides extra visual weight and identifies the primary action in a set of buttons -->
<button type="button" class="btn btn-primary">Primary</button>

<!-- Indicates a successful or positive action -->
<button type="button" class="btn btn-success">Success</button>

<!-- Contextual button for informational alert messages -->
<button type="button" class="btn btn-info">Info</button>

<!-- Indicates caution should be taken with this action -->
<button type="button" class="btn btn-warning">Warning</button>

<!-- Indicates a dangerous or potentially negative action -->
<button type="button" class="btn btn-danger">Danger</button>
```

It is important to note that the classes used above (ie: ".btn", ".btn-primary", "btn-success", etc) can also be used on the follow of elements:

- **hyperlinks:** Link
- **button elements:** <button class="btn btn-default" type="submit">Button</button>

- **button elements:** <button class="btn btn-default" type="submit">Button</button>
- **input type="button" elements:** <input class="btn btn-default" type="button" value="Input">
- **input type="submit" elements:** <input class="btn btn-default" type="submit" value="Submit">

Button Sizes

While the buttons rendered above look good and match Bootstrap's default style, we don't necessarily always want to render that size. To overcome this and add some flexibility to the sizing, Bootstrap has also provided the following sizing classes to wo buttons:


```
<p>
  <button type="button" class="btn btn-primary btn-lg">Large button</button>
  <button type="button" class="btn btn-default btn-lg">Large button</button>
</p>
<p>
  <button type="button" class="btn btn-primary">Default button</button>
  <button type="button" class="btn btn-default">Default button</button>
</p>
<p>
  <button type="button" class="btn btn-primary btn-sm">Small button</button>
  <button type="button" class="btn btn-default btn-sm">Small button</button>
</p>
<p>
  <button type="button" class="btn btn-primary btn-xs">Extra small button</button>
  <button type="button" class="btn btn-default btn-xs">Extra small button</button>
</p>
```

Dropdown Buttons

There are a few more interesting things that we can do to work with buttons (ie: setting "active" state, "disabled" state & creati level buttons), however one of the coolest (and most useful) button treatments that Bootstrap provides is the "dropdown butt


```
<div class="dropdown">
  <button class="btn btn-primary dropdown-toggle" type="button" id="dropdownMenu1" data-toggle
    Dropdown  <span class="caret"></span>
  </button>
  <ul class="dropdown-menu" aria-labelledby="dropdownMenu1">
    <li><a href="#">Action</a></li>
    <li><a href="#">Another action</a></li>
    <li><a href="#">Something else here</a></li>
    <li role="separator" class="divider"></li>
    <li><a href="#">Separated link</a></li>
  </ul>
</div>
```

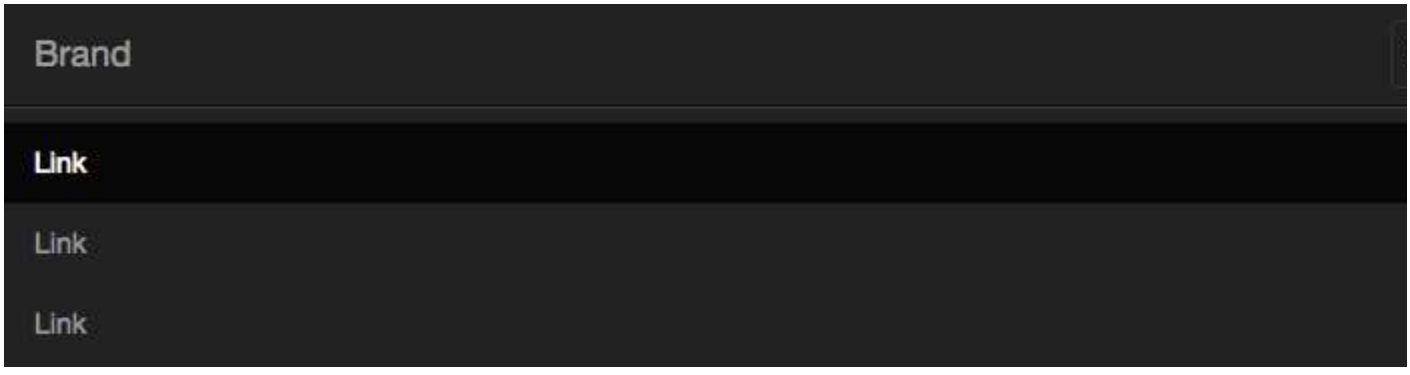
Navigation Bar

Almost every website you visit or web app you use will feature some sort of **navigation bar**. Users depend on this to navigate your app and explore all of the features/information available. Bootstrap has it's own responsive navigation bar that is highly customizable and works very nicely on mobile devices. To get started, let's create a navigation bar with a "Brand" (space for a l three (3) navigation links, the first of which is "active" (selected) - this would represent the current page / view:

Full Navigation Bar

Mobile (Compressed) Navigation Bar

Mobile (Expanded) Navigation Bar



```
<nav class="navbar navbar-inverse navbar-static-top">
  <div class="container">
    <!-- Brand and toggle get grouped for better mobile display -->
    <div class="navbar-header">
      <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">Brand</a>
    </div>

    <!-- Collect the nav links, forms, and other content for toggling -->
    <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
      <ul class="nav navbar-nav">
        <li class="active"><a href="#">Link</a></li>
        <li><a href="#">Link</a></li>
        <li><a href="#">Link</a></li>
      </ul>
    </div>
  </div>
</nav>
```

There's a lot going on in the above code, but a large chunk of it is boilerplate and is rarely changed. The main areas that we wo typically alter in the above code are:

```
<nav class="navbar navbar-inverse navbar-static-top">
```

Here, we have a few options on how the overall navigation bar will appear by changing which classes we include. We cannot ch the "navbar" class, however we can use the following other options:

- **navbar-inverse** can instead be: **"navbar-default"** - this will change the scheme from dark to light
- **navbar-static-top** can be either **removed** (resulting in rounded corners), **changed to "navbar-fixed-top"** which will alw the navbar in place at the **top of the page**, regardless of scroll position, or **changed to "navbar-fixed-bottom"** which w keep the navbar in place at the **bottom of the page**, regardless of scroll position

```
<a class="navbar-brand" href="#">Brand</a>
```

Next, we can skip down to the "navbar-brand" (unless you wish to change the id's from "bs-example-navbar-collapse-1" - in wh

simply do a find/replace). We do not typically change anything fundamental about this code except:

- **href="#"** would typically redirect back to the root ("/") or homepage of the website / application
- **Brand** - this text would usually be replaced with a logo ("brand") image.

```
<ul class="nav navbar-nav"> ... </ul>
```

The above unordered list simply contains the list of links that are available in the navigation bar. We can have one (1) or more lists and they can either be left-aligned (by default / **adding the class "navbar-left"**) or right-aligned (by **adding the class "navb right"**).

If we wish to add more or less links, we can add/remove them here. Additionally, we can add things like:

Dropdown Lists:

```
<ul class="nav navbar-nav">
    <li class="dropdown">
        <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup
        <ul class="dropdown-menu">
            <li><a href="#">Action</a></li>
            <li><a href="#">Another action</a></li>
        </ul>
    </li>
</ul>
```

Form Elements:

```
<ul class="nav navbar-nav">
   ...
</ul>
<form class="navbar-form navbar-right">
    <div class="input-group">
        <input type="text" class="form-control" placeholder="Search">
        <span class="input-group-btn">
        <button type="submit" class="btn btn-default">Submit</button>
        </span>
    </div>
</form>
```

Forms

Since our WEB322 app has been making extensive use of the Bootstrap form classes, we will be sticking with a simple exampl

Since our WEB322 app has been making extensive use of the Bootstrap form classes, we will be sticking with a simple example more in-depth description of the Bootstrap form classes, refer to the official documentation here: http://getbootstrap.com/css/#forms.

To get started using Bootstrap forms, you really only need to remember two classes: **form-group** and **form-control**. From the Bootstrap documentation:

> "Individual form controls automatically receive some global styling. All textual <input>, <textarea>, and <select> elements with **.form-control** are set to **width: 100%;** by default. Wrap labels and controls in **.form-group** for optimum spacing."


```
<form>
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" placeholder="Email">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1" placeholder="Passwo
  </div>
  <div class="form-group">
    <label for="exampleInputFile">File input</label>
    <input type="file" id="exampleInputFile">
    <p class="help-block">Example block-level help text here.</p>
  </div>
  <div class="checkbox">
    <label>
      <input type="checkbox"> Check me out
    </label>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

Bootstrap JavaScript (jQuery) Components

Due to time constraints, it is impossible to discuss all of the fantastic Bootstrap JavaScript Components and how they work in d
However, we will provide some examples for the more interesting/useful ones. If you are seriously interested in using Bootstra
your projects, the above link is a "must-read". Please note that like the other Bootstrap components, the code used below is la
boilerplate and there is little room for configuration out of the box - simply follow the pattern of elements and CSS classes and
Bootstrap framework will take care of the rest.

Dismissible Alerts

Dismissible Alerts in Bootstrap are simply small divs that provide a temporary message to the user, ie: "Warning: your session
out in 2 minutes". We often do not want to clutter the user interface with these alerts, so Bootstrap has included functionality
users to "dissmiss" the alert (by pressing a close ("x") button). Additionally alerts can be given a different colour depending on
of alert, including: red ("alert-danger"), yellow ("alert-warning"), blue ("alert-info") and green ("alert-success"):


```
<div class="alert alert-danger alert-dismissible fade in" role="alert">
    <button type="button" class="close" data-dismiss="alert" aria-label="Close">
        <span aria-hidden="true">×</span>
    </button>
    <strong>Error:</strong> Something went wrong.
</div>

<div class="alert alert-warning alert-dismissible fade in" role="alert">
    <button type="button" class="close" data-dismiss="alert" aria-label="Close">
        <span aria-hidden="true">×</span>
    </button>
    <strong>Warning:</strong> Something might go wrong soon.
</div>

<div class="alert alert-info alert-dismissible fade in" role="alert">
    <button type="button" class="close" data-dismiss="alert" aria-label="Close">
        <span aria-hidden="true">×</span>
    </button>
```

```
    <strong>Information:</strong> Something is happening.
</div>

<div class="alert alert-success alert-dismissible fade in" role="alert">
    <button type="button" class="close" data-dismiss="alert" aria-label="Close">
        <span aria-hidden="true">×</span>
    </button>
    <strong>Success:</strong> Something went right!
</div>
```

Tabs

Tabs are an extremely common user-interface component. They have been used in these notes (see Week 7 - "Putting it All To and play a significant role in optimizing space on a screen for categorized information. Using jQuery, the Bootstrap framework created a standard HTML pattern that we can leverage to create a functioning tab control without writing a single line of JavaS

Once again the following code is largely boilerplate out of the box. As long as we follow the predefined structure, our tabs will properly.


```
<!-- Nav tabs -->
<ul class="nav nav-tabs" role="tablist">
    <li role="presentation" class="active"><a href="#home" aria-controls="home" role="tab" dat
    <li role="presentation"><a href="#profile" aria-controls="profile" role="tab" data-toggle=
    <li role="presentation"><a href="#messages" aria-controls="messages" role="tab" data-toggl
    <li role="presentation"><a href="#settings" aria-controls="settings" role="tab" data-toggl
</ul>

<!-- Tab panes -->
<div class="tab-content">
    <div role="tabpanel" class="tab-pane active" id="home"><br />home content</div>
    <div role="tabpanel" class="tab-pane" id="profile"><br />profile content</div>
    <div role="tabpanel" class="tab-pane" id="messages"><br />messages content</div>
    <div role="tabpanel" class="tab-pane" id="settings"><br />settings content</div>
</div>
```

In the above code, notice how we have some identifiers repeated across the "Nav tabs" section and the "Tab panes" section? T primarily the **href** and **aria-controls** attributes in the "Nav tabs" section. The **href** attributes each link to the **corresponding "T id** that they wish to show when clicked, and the **aria-controls** attribute helps aid in the accessibility of the control.

TAB CONFIGURATION

Even though the tabs are fairly standard, we do have some configuration options available, such as:

- **Using a Fade Effect**: To make tabs fade in, add the class "fade" to each "tab-pane". The first tab pane must also have th class to make the initial content visible.

```
<div class="tab-content">
    <div role="tabpanel" class="tab-pane fade in active" id="home"><br />home content</di
    <div role="tabpanel" class="tab-pane fade" id="profile"><br />profile content</div>
    <div role="tabpanel" class="tab-pane fade" id="messages"><br />messages content</div>
    <div role="tabpanel" class="tab-pane fade" id="settings"><br />settings content</div>
</div>
```

- **Adding "Pill" Styling:** We can make the tabs appear as buttons by adding the class "nav-pills" to the "Nav tabs" ele

```
<ul class="nav nav-tabs nav-pills" role="tablist">
```

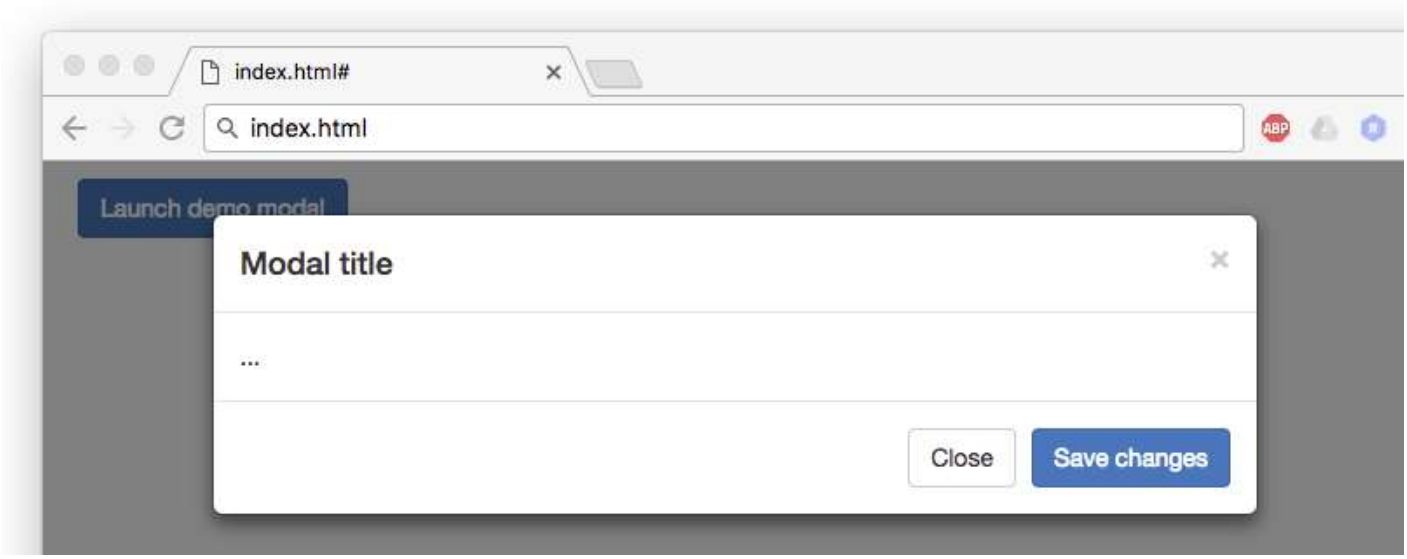
- **Stacking the "Pill" Tabs:** Another option is to display the tabs above one another in a "stack". Please note, this the **not** thing as "vertical tabs". Stacking the tabs simply places each tab "pill" in a vertical stack with the pane(s) at the bottom.

```
<ul class="nav nav-tabs nav-pills nav-stacked" role="tablist">
```

Modal Window

The "modal window" is one of the most important components in the list and you will find yourself needing it on every project Essentially, a modal window is a custom in-page popup window that blocks the background content from being clicked on / in with. You will often see login/registration forms, chat windows, forms to edit table row data, etc. placed in modal windows.

The Bootstrap implementation is very clean and easy to use - it also has the bonus of ensuring that the generated modal winc "responsive" and will not break the view or cause excessive scrolling when accessed on a mobile device. The following code is example of how a modal window is defined and how it can be "wired up" to be opened by clicking a button.



```
<!-- Button trigger modal -->
<button type="button" class="btn btn-primary" data-toggle="modal" data-target="#myModal">
  Launch demo modal
</button>

<!-- Modal -->
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria
        <h4 class="modal-title" id="myModalLabel">Modal title</h4>
      </div>
      <div class="modal-body">
        ...
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary" onclick="console.log('saved!'); $('#myMo
      </div>
    </div>
  </div>
</div>
```

Once again, there's a lot going on in the above code, but (as before) a large chunk of it is boilerplate and is rarely changed. The areas that we would typically alter in the above code are:

```
<button type="button" class="btn btn-primary" data-toggle="modal" data-target="#myModal">
  Launch demo modal
</button>
```

This is simply the element that actually launches the modal. This could be **any element** that has the properties **data-toggle="r** and **data-target="#someId"** where "#someId" will be the the id of your "modal" ... .

```
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel
```

The only things that we can really change here are the **id** of the "modal" and the **aria-labelledby** value (this corresponds to the your "modal-title" <h4> - see below)

```
<h4 class="modal-title" id="myModalLabel">Modal title</h4>
```

This is the text that appears as the "title" of the modal window. Here we would typically change the **inner text** and the **id** (we ju
to make sure that the id matches the "aria-labeledby" property above.

```
<div class="modal-body">
        ...
</div>
```

The "modal-body" element is where we will place the content of the modal window. This could be anything, however typically
rows/colums are placed here (see "Responsive Grid System" above) to position the content.

```
<div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary" onclick="console.log('saved!'); $('#myMo
</div>
```

Finally we have the "modal-footer". Once again, we can have anything we like in this element, however it is common to have a
button (to cancel the action using the data-dismiss="modal" property) and a "Save" or "Submit" button (to confirm the action a
programatically hide the modal using **$("#modalId").modal("hide");**, where "#modalId" is the id of the modal window).
