Intro to JavaScript

Jen Zajac

Why do we want to use JS?

- HTML – content
- CSS – presentation
- JavaScript – behavior

A brief history of JS

- Created by one person in 1995
- Originally called Mocha, renamed to JavaScript for marketing reasons
- Has actually nothing in common with Java
- Slow going with improvements over the years, but now up to version 5 of ‘ECMAScript’
- Version 6 (aka 2015) exists but is not yet supported by most browsers

Our focus today

- “Client-side” JS
- Build a simple page
- Use a very common JavaScript library – jQuery
- Why use a library?
  - Different browser implementations
  - Easier to write than most ‘vanilla js’

We we’re building today

- Use govt.nz consultations API
- Build an interactive table for the upcoming government consulations
- Will base our project on a popular basic project template – HTML5Boilerplate
- Here’s one I prepared earlier:

https://github.com/jenofdoom/summer-of-tech-intro-js

Getting set up

- Follow through the setup instructions on that page
- Remember – all of the coding we’ll be doing is jQuery specific – if you need to look something up google jQuery’s api docs or use http://jqapi.com/

Getting started

- All of our JavaScript is loaded via the script tag at the bottom of ‘index.html’
- We will do all our scripting in ‘js/main.js’
- All our code should go inside $(document).ready(function() {

AJAX request

- ‘Asynchronous JavaScript and XML’
- We need to do a GET request
- We want to include some parameters
- Remember: in ‘main.js’, inside the document ready block

```
var url =
"https://www.govt.nz/api/v2/consultation/list";

$.getJSON(
    url,
    {
        limit: 'all',
        status: 'current',
        sort: 'end'
    }
)
.done(function(data) {
    console.log(data);
})
.fail(function(error) {
    console.log("Request Failed:", error);
});
```

Checking that that worked

- In a browser, open ‘index.html’
- Open the web developer tools for your browser – press F12 or right click on the page and choose ‘Inspect element’
- In your web dev tools, navigate to the ‘Console’ tab
- When you reload the page you should see log messages in the console

Displaying the info

- We want to display that data in a table on the page
- We’ll need to build the HTML elements that we need for a table in our JavaScript
- We also need a container element in our ‘index.html’ files, which the table will be inserted into

In ‘index.html’

```
<h1>Upcoming Government
Consultations</h1>

<div id="app"></div>
```

In ‘main.js’

Change:

```
.done(function(data) {
  console.log(data);
})
```

to:

```
.done(function(data) {
  buildTable(data.consultations);
})
```

Underneath the AJAX $.get function:

```
var buildTable = function(consults) {
  var table = $('<table />');
  var tbody = $('<tbody />');

  $(consults).each(function(index, consult) {
    var row = $('<tr />');
    var title = $('<td />').text(consult.title);

    row.append(title);
    tbody.append(row);
  });

  table.append(tbody);
  $('#app').append(table);
};
```

Now we have an ugly table :)

- Let’s tidy it up a bit
- Need to add:
  - Column heading
  - Some Bootstrap (CSS library) class
- Let’s make the following changes (new bits are in white, old bits in grey):

In the buildTable function:

```
var table = $('<table />');
var tbody = $('<tbody />');
var thead = $('<thead />');
var theadRow = $('<tr
/>').append('<th>Title</th>');
```

Near the bottom:

```
thead.append(theadRow);
table.addClass('table table-hover');
table.append(thead, tbody);
```
~~table.append(tbody);~~
```
$('#app').append(table);
```

Better! But we could use some more columns

- Start date
- End date

At the top of the buildTable function:

```
var thead = $('<thead />');
~~var theadRow = $('<tr />').append('<th>Title</th>');~~
var theadRow = $('<tr
/>').append('<th>Title</th>','<th>Start</th>',
'<th>End</th>');
```

In the row loop:

```
var title = $('<td />').text(consult.title);
var startDate = $('<td />').text(consult.start);
var endDate = $('<td />').text(consult.end);

~~row.append(title);~~
row.append(title, startDate, endDate);
```

Hmmm...

- 2016-07-11T00:00:00+12:00 is not a super user friendly date
- Let’s add a new function to fix that
- Adding it as a function means it will be nicely reuseable

Underneath the buildTable function:

```
var dateFormatter =
function(dateString){
    var date = new Date(dateString);
    var day = date.getDate();
    var month = date.getMonth();
    var year = date.getFullYear();

    date = day +  '/' + month + '/' +
year;
    return date;
};
```

In the row loop in buildTable:

```
~~var startDate = $('<td />').text(consult.start);~~
var startDate = $('<td />').text(dateFormatter(consult.start));
~~var endDate = $('<td />').text(consult.end);~~
var endDate = $('<td />').text(dateFormatter(consult.end));
```

What if we want to zerofill the dates?

- e.g. show 06/06/2016 not 6/6/2016
- we can use a ternary operator to achieve this
- the format of a ternary operator is as follows:

  ```
  test ? resultIfTrue : resultIfFalse
  ```

In the dateFormatter function:

```
day = day < 10 ? '0' + day : day;
month = month < 10 ? '0'  +  month :
month;
date = day + '/' + month + '/' +
year;
return date;
```

Adding the topics

- The topics are an array in the data structure
- Arrays look like:

  [‘something’, 4, ‘someotherthing’]

- We should iterate over the array in case there is more than one topic per consultation

In the row loop in buildTable:

```
var endDate = $('<td
/>').text(dateFormatter(consult.end));
var topics = $('<td />');

$(consult.topic).each(function(index, topic) {
  var topicSpan = $('<span />').text(topic);

  topicSpan.addClass('label label-default');
  topics.append(topicSpan);
});

~~row.append(title, startDate, endDate);~~
row.append(title, startDate, endDate, topics);
```

At the top of the buildTable function:

```
~~var theadRow = $('<tr
/>').append('<th>Title</th>','<th>Sta
rt</th>', '<th>End</th>');~~
var theadRow = $('<tr
/>').append('<th>Title</th>','<th>Sta
rt</th>', '<th>End</th>',
'<th>Topics</th>');
```

Adding the description

- There isn’t really enough room in the table row for the consultation description, because it’s usually at least a paragraph of text
- Instead, we can add a description row immediately after each consulation, which we will toggle show/hide of on click
- The description has HTML markup in it so we need to use .html() not .text()

In the row loop in buildTable:

```
var topics = $('<td />');
var descriptionRow = $('<tr
/>').addClass('hidden');
var description = $('<td colspan="4"
/>').html(consult.description);
var moreInfo = $('<a />').text('Find out
more').attr('href',
consult.url).addClass('btn btn-default');
```

At the bottom of the row loop in buildTable:

```
row.append(title, startDate, endDate,
topics);
row.addClass('clickable');
description.append(moreInfo);
descriptionRow.append(description);
~~tbody.append(row);~~
tbody.append(row, descriptionRow);
```

At the bottom of the row loop in buildTable:

```
row.append(title, startDate, endDate,
topics);
row.addClass('clickable');
description.append(moreInfo);
descriptionRow.append(description);
~~tbody.append(row);~~
tbody.append(row, descriptionRow);
```

In css/main.css:

```css
.clickable {
   cursor: pointer;
}

table tr td:first-child {
   width: 60%;
}

.text-middle {
   text-align: center;
}

.hidden {
   display: none !important;
   visibility: hidden !important;
}
```

It’s there but we can’t yet see it...

- If you reload the page and use the web devloper tools to inspect the table, you can see the hidden rows (hidden with the css class ‘hidden’)
- How do we make them visible on click?
- We need to add a click event – and we can’t set it up until the element exists

At the bottom of the buildTable function:

```
$('#app').append(table);

$('table tr.clickable').on('click',
  function(event){
    $(this).next().toggleClass('hidden');
});
```

At the bottom of the buildTable function:

```
$('#app').append(table);

$('table tr.clickable').on('click',
  function(event){
    $(this).next().toggleClass('hidden');
});
```

At the bottom of the buildTable function:

```
$('#app').append(table);

$('table tr.clickable').on('click',
  function(event){
    $(this).next().toggleClass('hidden');
});
```

That should work now!

- This method uses jQuery ‘on’ event binding to just add a class to ‘display: none’ the element – this has better performance than removing/adding the element every time
- Note: this JavaScript show/hide is not very accessible right now – to do it properly we should add some keyboard functionality and ARIA attributes

For our last trick

- Let’s add a search box so we can filter the list
- We’ll need to add an input element to the HTML to catch the search term
- After that, we need to add another event handler that will watch the input for keypresses

In index.html, after the H1:

```
<h1>Upcoming Government Consultations</h1>

<div class="form-group">
  <label for="search">Search</label>
  <input
    id="search"
    type="search"
    class="form-control"
    placeholder="Search consultation titles"
    autocomplete="off"
  >
</div>

<div id="app"></div>
```

In index.html, after the H1:

```html
<h1>Upcoming Government Consultations</h1>

<div class="form-group">
  <label for="search">Search</label>
  <input
    id="search"
    type="search"
    class="form-control"
    placeholder="Search consultation titles"
    autocomplete="off"
  >
</div>

<div id="app"></div>
```

At the bottom of the buildTable function:

```
$('table tr.clickable').on('click', function(event){
    $(this).next().toggleClass('hidden');
});

$('#search').on('keyup', function(event){
  var input = $(this).val().toLowerCase();

  $('table tbody tr.clickable').each(function(index, row) {
    var titleCell = $(row).children()[0];
    var titleText = $(titleCell).text().toLowerCase();

    if (titleText.indexOf(input) === -1) {
      $(row).addClass('hidden');
      $(row).next().addClass('hidden');
    } else {
      $(row).removeClass('hidden');
    }
  });
});
```

Actually, one last thing

- Let’s put in some text to indicate that the table data is loading while we wait for the API call to complete

In index.html, before the app div:

```
<p id="loading" class="text-
middle">Loading</p>

<div id="app"></div>
```

In main.js, in buildTable, before the event handlers:

```
table.append(thead, tbody);
$('#loading').addClass('hidden');
$('#app').append(table);
```

Success!

- Our web app is complete
- We should be able to search by title, read the start and close dates, and display the description with a link to more information

A different method

- Building HTML with jQuery is fiddly and it becomes hard to maintain
- A better approach would be a templating language, or perhaps a JavaScript framework
- If you are using git, commit your changes and run ‘git checkout vue-example’ to see the same app built in Vue, a lightweight JS framework

Ideas for enhancements

- Add a topic selector control at the top of the page that filters the list to only show a given topic
- Add a ‘closing soon’ warning to any consultations that are due to close in the next 30 days
- Add ARIA attributes and keyboard controls to the description display for better accessibility

Other things to consider

- For a production site, you should minify your javascript (and preferably gzip it too)
- The way our app works, we are cluttering up the ‘global scope’ - namespacing our code would be better
- Code comments are great – prepend your comment with //

Where to from here?

- JS Masterclass
- Follow the tutorial for a framework like Vue, React or Angular
- Learn about package management with bower or npm
- Learn about server-side JavaScript with node.js
- Try and build something!