VUE.JS

INTRODUCTION

Vue.js is a Javascript library whose primary focus is on the 'View' of MVC architecture. It is great for connecting data to the DOM.

Unlike some other Javascript frameworks (Angular, React, Ember), Vue.js has little configuration and offers quick solutions to common problems – ie connecting data to the DOM.

It is particularly useful for integrating data into web application via Javascript rather than via a server side script. This is primarily achieved via AJAX. As Vue.js itself focuses on placing values into the DOM it leaves the AJAX functionality to others. As such you could use jQuery or a light-weight AJAX library like AXIO.

GET THE FILES

You will need to reference the Vue.js file into your web application in the same way you work with the likes of jQuery. As with jQuery you can choose to host the file yourself or use a CDN (Content Distribution Network).

A CDN version is hosted on cdn.jsdelivr.net at:

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

Alternatively, the lab files include a local copy in *js/vue.min.js*.

To use Vue.js add a `<script>` tag to the `<head>` of your HTML file. The code we'll write can be added to the bottom of a HTML file generally before the closing `</body>` tag. This will be done so that the DOM is loaded and understood.

DATA BINDING

Open file *data_binding.html*. Add the Vue.js file as outlined above.

Then add a `<div>` as follows to the body ie:

```
<div id="app">
      <h1></h1>
</div>
```

Now add a `<script>` tag just before the closing `</body>` tag.

Here we'll create a new Vue object. This takes an object as a parameter. The two properties of this object we'll use most frequently are:

el: The mounting element that will receive the data. This is specified with a CSS selector
data: The data to pass to the element outlined above.

For example, we could add:

```
var vm = new Vue({
 el: '#app',
  data: {
    welcomeMessage: 'Hello Vue!'
  }
})
```

Here the `el` is an `ID` selector of `#app`. In data we have a value of `welcomeMessage`.

Add mount the data to the el use `{{..}}`. Change the HTML to:

```
<div id="app">
      <h1>{{welcomeMessage}}</h1>
</div>
```

Save and preview your file.

The use of `{{..}}` is known as string interpolation. Notice that when using this technique there is no need to reference `data.welcomeMessage`. The process of binding the data to the element means any values declared in data will be available to the element.

Try adding a second value to the data of `pet` with a value of `dog` and add it to the DOM.

DATA BINDING ATTRIBUTES

Data binding with the {{.. }} works great for text but we'll also want to bind other types of data such as attributes values for hyperlinks or images.

Assuming you have declared a URL in a data value of `myLink` you might expect to be able to do:

```
<a href="{{myLink}}">Link to SHU</a>
```

However, this doesn't not work. If binding attribute values use the prefix `v-bind:` before the attribute eg:

```
<a v-bind:href="myLink">Link to SHU</a>
```

When using `v-bind:` there is no need to use the `{{..}}` around the value itself.

As this is such a common construct you can use the shorthand of just a `:` (colon) eg:

```
<a :href="myLink">Link to SHU</a>
```

The value inside the attribute when bound in this fashion can be any Javascript expression. As such you can concatenate strings eg:

```
 <img v-bind:src="'images/'+myImage">
```

or with the shorthand:

```
<img :src="'images/'+myImage">
```

The custom attributes such as `v-bind` are known as directives.

Amend your *data_binding.html* file by adding a hyperlink and an image using dynamic values.

Your HTML may appear as:

```
<!--Long Hand-->
<p><a v-bind:href="myLink">Link to SHU</a></p>
<!--Short Hand-->
<p><a :href="myLink">Link to SHU</a></p>
<p><img v-bind:src="'images/'+myImage"></p>
```

The Vue model amended as:

```
var vm = new Vue({
 el: '#app',
  data: {
    welcomeMessage: "Hello Vue!",
      pet: "Dog",
      myLink: "http://www.shu.ac.uk",
      myImage: "matrix.jpg"
  }
})
```

TWO WAY DATA BINDING WITH V-MODEL

Two way data binding allow the view to update the model value and vice-versa. To illustrate this open the sample file *two-way-binding.html*.

Notice that this has an input text form field and HTML that reference a Vue model object declared in the script tag at the bottom of the page:

```
<script>
var vm = new Vue({
 el: '#app',
  data: {
    myEmail: "m.j.cooper@shu.ac.uk"
  }
})
</script>
```

Using data binding we can view the value of myEmail with:

```
<p>{{myEmail}}</p>
```

We can also two way bind this value to that in the input form by adding the directive `v-model` eg:

```
<input type="email" name="userEmail" v-model="myEmail">
```

Save and preview your file. When changing the form value the view should immediately update the value.

This is two way binding and by default it is triggered by changing the value. We can change that event however, by using the directive `v-model.lazy` such that the value is only updated onblur eg:

```
<input type="email" name="userEmail" v-model.lazy="myEmail">
```

The likes of lazy is known as a modifier.

BINDING COMPLEX DATA

As well as simple strings and numbers, arrays can be sent to the view via data bindings.

Open the file *arrays-objects.html*. Add the following to create Vue model:

```
var vm = new Vue({
 el: '#app',
  data: {
      genres : ['horror','comedy','drama','action']
  }
})
```

This could be used to create a loop with the `v-for` directive. The result is in effect a for/in loop that loops the properties of an object or the values of an array.

```
<ul>
      <li v-for="genre in genres">
            {{ genre }}
      </li>
</ul>
```

Add an object to the data in the Vue model as follows:

```
cinema : {location : 'Sheffield', screens: 12, opening: 'Mon-Sun', prices:
'from &pound;2.00'}
```

This can then be looped as follows:

```
<ul>
      <li v-for="(value, key) in cinema">
      {{ key }}: {{ value }}
      </li>
</ul>
```

Notice that the HTML entity is treated as text. Data binding in default is done as plain text. If you wish to bind HTML (and that includes HTML entities) then use the `v-html` directive.

Amend the loop to use `v-html` as follows:

```
      {{ key }}: <span v-html="value"></span>
```

We can also have arrays of objects. Add an array of objects to the data as follows:

```
films : [
      {filmName: 'Skyfall', certificate: 12},
      {filmName: 'Star Wars', certificate: 'PG'},
      {filmName: 'Wizard of Oz',certificate: 'U'},
      {filmName: 'Toy Story', certificate: 'U'},
      {filmName: 'Omen', certificate: 18},
]
```

This could be looped with:

```
<ul>
      <li v-for="film in films">
      {{ film.filmName }} Certificate: {{ film.certificate }}
      </li>
</ul>
```

CONDITIONAL LOGIC

As well as loops, there are also directives for conditional logic.

Open the file *conditional-logic.html*. Notice that this already includes a Vue model with a range of values declared.

```
  data: {
            active : true,
            price : 10,
            films : [
                  {filmName: 'Skyfall',certificate: 12},
                  {filmName: 'Toy Story',certificate: 'U'},
                  {filmName: 'Star Wars',certificate: 'PG'},
                  {filmName: 'Wizard of Oz',certificate: 'U'},
                  {filmName: 'Omen',certificate: 18},
            ]
  }
```

We'll use these values to investigate conditional logic.

The `v-show` directive toggles the CSS display property of an element.

In which case content can be set to display based on the `active` value declared.

```
<p v-show="active">Show This</p>
```

Javascript expressions can also be used:

```
<p v-show="price == 10">Price is &pound;10.00</p>
```

In the console, you could hide this by typing:

```
vm.price = 12
```

This can also be applied along with the `v-for` directive eg:

```
<ul>
      <li v-for="film in films" v-show="film.certificate == 'U'">
            {{ film.filmName }} Certificate: {{ film.certificate }}
      </li>
</ul>
```

We could also use the `v-if` and `v-else` directives.

```
<p v-if="price > 5">Expensive</p>
<p v-else>Cheap</p>
```

The `v-if` and `v-else` directives do re-write the DOM as opposed to just toggle the CSS display which is what the `v-show` directive does.

EVENTS AND METHODS

Open the file *events.html* to add the following events.

Events in Vue.js are added inline with directives eg:

```
<button v-on:click="console.info('hi')">Just a Console Call</button>
```

Code is better structured when the events are used to call a method. Methods can be declared as part of the Vue model as follows:

```
var vm = new Vue({
 el: '#app',
 methods: {
        myEvent : function(){
              console.info('myEv1')
        }
  }
})
```

They are then called inline eg:

```
<p><a href="#" v-on:click="myEvent">Hyperlink calling myEvent</a></p>
```

Multiple methods can be declared.

```
var vm = new Vue({
 el: '#app',
   methods: {
        myEvent : function(){
              console.info('myEv1')
        },
        myEventPara : function(someVal){
              console.info(someVal)
        }
  }
})
```

Parameters can be passed to event handlers using the method call.

```
<a href="#" v-on:click="myEventPara('someVal')">Send Parameter</a>
```

These calls can also send the event object itself via $event.

```
<a href="#" v-on:click="myEventWithObject($event, 'someVal')">Send
Parameter and event</a>
```

Add a method as follows:

```
myEventWithObject : function(ev someVal){
            ev.preventDefault();
            console.info(someVal);
        }
```

As with binding there is a shorthand for `v-on:`. It can be replaced with an `@`.

```
 <a href="#" @click="myEvent">Hyperlink calling myEvent2</a>
```

You may have noticed that Vue.js is using inline events that are generally frowned upon in Javascript development. The Vue.js documentation defends this practice by stating the following:

1. It's easier to locate the handler function implementations within your JS code by skimming the HTML template.
2. Since you don't have to manually attach event listeners in JS, your ViewModel code can be pure logic and DOM-free. This makes it easier to test.
3. When a ViewModel is destroyed, all event listeners are automatically removed. You don't need to worry about cleaning it up yourself.

GETTING DATA WITH AJAX

Setting up the endpoints

Before experimenting with AJAX and Vue.js we need to set up some endpoints. In the data folder of the starter files there are two endpoint files *allFilms.php* and *searchFilms.php*. These both reference a connection file. You will need to edit this to match your database settings.

Loading data via AJAX is a key reason for using Vue.js as it makes binding the data to the DOM very straightforward. However, Vue.js does not itself come with AJAX support. As such you will need to use another library such as jQuery or Axios.

If the AJAX data is to be loaded on a page load then we need to consider the Vue.js life cycle so that the AJAX call is made at the most appropriate time.

Vue.js works by creating a 'virtual DOM' which is then 'mounts' to a target element `el`. The two key events here are `created` and `mounted`.

- The `created` event is called when Vue element has been created and has all the reactive data bindings set up.
- The `mounted` event is triggered when the DOM has been rendered based on the Vue element created above. At this stage the DOM can be manipulated.

Therefore, to load data on page most efficiently we can start the AJAX call on `created` as the data bindings are in place. When the `mounted` event triggers, which will be shortly afterwards the `created` event, then the data will then appear in the DOM.

Open the file *jQuery-all-films.html*. Notice that this already has jQuery and Vue.js loaded. Add a Vue model as follows:

```
var vm = new Vue({
    el: '#app',
    data: {
       films: []
    },
    created: function () {
      $.getJSON('data/allFilms.php', function(data){
            console.dir(data);
            vm.films = data;
        });
    }
});
```

In the body of the HTML add the following HTML to build a loop around the data:

```
<main id="app">
      <div class="imgGrid">
      <div class="grid" v-for="film in films">
      <img v-bind:src="'images/' + film.filmImage" alt="">
       <h2>{{film.filmName}}</h2>
     </div>
      </div>
</main>
```

AXIOS

As much of the functionality of jQuery is done by Vue.js then it makes sense to use a lighter AJAX focused library such as Axios.

Axios can be downloaded from Github (https://github.com/axios/axios).

Save a copy of your jQuery example above and rename as *axios-all-films.html*.

Edit this new file to create an axios version of the above with the following AJAX call:

```
var vm = new Vue({
    el: '#app',
    data: {
       films: []
    },
    created: function () {
        axios.get('data/allFilms.php')
      .then(response => {
            // JSON responses are automatically parsed.
            this.films = response.data
      })
      .catch(e => {
            this.errors.push(e)
      })
    }
});
```

AJAX CALLS BY EVENT

AJAX calls can also be triggered with events such as a search form.

Open the file *jQuery-search.html*. This includes links to jQuery and Vue.js.

Create a two way binding with v-model eg:

```
<label for="searchTerm">Search:
      <input type="search" v-model="searchTerm" v-on:change="searchFilms">
</label>
```

We could also add some conditional logic as follows:

```
    <p v-if="searchTerm">Search Term: {{searchTerm}}</p>
```

To create a search page that calls an endpoint via jQuery we use:

```
var vm = new Vue({
    el: '#app',
    data: {
       films: [],
         searchTerm : ""
    },
    methods: {
      searchFilms : function () {
            var sendVars = {"searchTerm" : vm.searchTerm};
            console.info(sendVars);
            $.getJSON('data/searchFilms.php', sendVars, function(data){
                    vm.films = data;
                });
             }
      }
});
```

This would then need binding to the DOM with:

```
<div class="grid" v-for="film in films">
      <h2>{{film.filmName}}</h2>
      <img v-bind:src="'images/' + film.filmImage" alt="">
</div>
```

This loop could be placed in some conditional logic in case no results are found:

```
<div class="imgGrid" v-if="films.length > 0">
      <div class="grid" v-for="film in films">
      <h2>{{film.filmName}}</h2>
       <img v-bind:src="'images/' + film.filmImage" alt="">
      </div>
</div>
```

As in the previous example you could re-create this example using Axios rather than jQuery.

Save a copy of your *jquery-search.html* file and rename it as *axios-search.html*.

Edit this new file to create an axios version.

This could be achieved with Axios by amending the `searchFilms` method as follows:

```
searchFilms : function(){
      axios.get('data/searchFilms.php', {params: {"searchTerm" :
vm.searchTerm}})
            .then(response => {
            // JSON responses are automatically parsed.
            this.films = response.data
            })
            .catch(e => {
            this.errors.push(e)
            })
      }
```

COMPONENTS

Components allow developers to create custom tags that encompass a range of functionality.

Open the file *component-with-data.html*.

For example, we could create a custom element as follows:

```
<main id="app">
  <my-component :films="films"></my-component>
</main>
```

This element has been `v-bind` to the films variable.

We can create the component as follows:

```
Vue.component('my-component', {
  template: '<div class="imgGrid">\
            <div class="grid" v-for="film in films">\
                  <div class=""><h2>{{film.filmName}}</h2></div>\
            </div>\
      </div>',
  props : ['films']
})
```

Then build the Vue object with:

```
var vm = new Vue({
      el: '#app',
      data: {
                  films : []
      },
      created: function () {
        axios.get('data/allFilms.php')
      .then(response => {
            // JSON responses are automatically parsed.
            this.films = response.data
      })
      .catch(e => {
            this.errors.push(e)
      })
    }
});
```

THINGS TO DO:

- Expand the component example *component-with-data.html* to display film images.
- Create a new endpoint to display the details of one film only based on the filmID primary key.
- Use this endpoint to create a new 'details' page that can be linked to from the search page and that displays all the film information.
- Use either jQuery or Axios to load the data.