Lab 6: Javascript for the client and VueJS

1 JavaScript for the front end

Taken from: http://study.com/academy/lesson/what-is-javascript-function-uses-quiz.html

Where HTML and CSS allow web developers to format the content of a web page, JavaScript allows them to make the page dynamic. For example, HTML/CSS allows for making text bold, creating text boxes, and creating buttons, whereas JavaScript allows for changing text on the page, creating pop-up messages, and validating text in text boxes to make sure required fields have been filled.

JavaScript makes web pages more dynamic by allowing users to interact with web pages, click on elements, and change the pages.

1.1 Exercise 1: JavaScript form validation

1. Create a new HTML page that contains the following code. Note the method, action and name of the text box. These are the values that the form on the Google home page uses to interact with its server.

```
<!DOCTYPE html>
<html>
	<head>
		<title>Lab 6: Exercise 1</title>
	</head>
	<body>
		<form method="GET" action="https://www.google.com/search">
			<input type="text" name="q" id="search_string" />
			<input type="submit" value="Search" />
		</form>
	</body>
</html>
```

2. Open the HTML file in your browser and test. The form will query Google with the search string that is entered into the text box
3. In the opening form tag, add an attribute called ‘onsubmit’ that has a value of ‘return validateForm()’

```
<form method="GET" action="https://www.google.com/search" onsubmit="return
validateForm()">
```

4. Create a new file called ‘validate.js’

5. Inside your JavaScript file, write the validateForm() function. This function should check the value of the text field on the form. If the field is populated (!= “”) then the function should return true, else provide a popup message and return false.

```
function validateForm(){

	var search_string = document.getElementById("search_string").value;

	if(search_string == ""){
		alert("Search string is empty!");
		return false;
	}else{
		return true;
	}
}
```

6. In your HTML file, use the script tag to import your javascript file. Read here, to learn why we import javascript at the bottom of our web page: https://robertnyman.com/2008/04/23/where-to-include-javascript-files-in-a-document/

```
		</form>
		<script type="text/javascript" src="./validate.js"></script>
	</body>
```

7. Open your HTML file in the browser and test your form. The form will not submit to the server unless there is a query entered into the text box.

2 JavaScript Frameworks (VueJS)

Taken from: https://vuejs.org/v2/guide/

Vue (pronounced /vjuː/, like view) is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is very easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.

2.1 Exercise 2: Getting started

To use Vue, a script is imported into your project like any other JavaScript file. This can be done by downloading the file, using a package manager like npm or bower, or by using a CDN (https://en.wikipedia.org/wiki/Content_delivery_network). For simplicity, we will use a CDN as shown below:

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <title>My Application</title>
  </head>
  <body>

  <script src="https://unpkg.com/vue"></script>
  </body>
</html>
```

Now that we have Vue imported into our project, we can start building our app. We will start with a 'hello world' example.

1. Create a new file called 'app.js,' import this file into your HTML file **underneath** where you import Vue
2. Inside your app.js file, add the following code:

```
const app = new Vue({
 el: '#app',
 data: {
   message: 'Hello World!'
 }
})
```

What's happening?
Here we are initiating a new Vue instance into a variable called 'app.' This instance will bind to an element in the DOM (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction#What_is_the_DOM) through the 'el' field, in this case we are binding it to an element with an ID of 'app.' Once Vue is binded to this element, data can be shared between the instance and the DOM. The rest of this exercise shows an example of this two-way data binding.

3. Now, inside your HTML, create a div with an ID of 'app' (before you import your app.js file and Vue). Inside this div, print the contents of the message variable using double curly brackets:

```
<div id="app">
  {{ message }}
</div>
```

4. Run the application in the browser to test

2.2 Exercise 3: Vue directives

Directives are special attributes with the v- prefix. Directive attribute values are expected to be a single JavaScript expression (with the exception for v-for, which will be discussed later). A directive's job is to reactively apply side effects to the DOM when the value of its expression changes.

2.2.1 Exercise 3.1: Conditional statements in Vue

1. Make a copy of your solution to exercise 2.
2. In app.js, add another variable under message called ‘visible.’ Set its value to true.
3. In your HTML, put the displayed message into ‘h1’ tags. Use ‘v-if’ to make the message only displayed if ‘visible’ is true.

```
<div id="app">
  <h1 v-if="visible">{{ message }}</h1>
</div>
```

4. Test in your browser with visible set to true and false.

2.2.2 Exercise 3.2: Looping in Vue

1. Make a copy of your solution to exercise 3.2.
2. In app.js, add another variable under message called ‘shopping_list.’ Its value should be a list of items typically found in a shopping list. These list items should be JSON objects with a name and a price.

```
data: {
    message: 'Hello World!',
    shopping_list: [
        { name: 'bread', price: 2.75 },
        { name: 'milk' , price: 2.50 },
        { name: 'pasta', price: 1.99 }
    ]
}
```

3. In your HTML, add an ordered list that uses the ‘v-for’ directive for listing the names of the items in your shopping list:

```
<ol>
  <li v-for="item in shopping_list">
    {{ item.name }}
  </li>
</ol>
```

4. Open in your browser to test

2.3 Exercise 4: Vue methods

Inside a Vue instance, you can also define methods that are relevant to that particular instance. In this exercise, we will create a method for calculating the total of our shopping list from exercise 3.2.

1. Make a copy of exercise 3.2
2. In app.js, after the closing of the ‘data’ object, create a new object called ‘methods.’

```
const app = new Vue({
    el: '#app',
    data: {
        message: 'Hello World!',
        shopping_list: [
            { name: 'bread', price: 2.75 },
            { name: 'milk' , price: 2.50 },
            { name: 'pasta', price: 1.99 }
        ]
    },
    methods: {

    }
});
```

3. Create a function called ‘calculateTotal.’ This function will loop through the items of shopping_list and return a sum of all the prices.

```
const app = new Vue({
    el: '#app',
    data: {
        message: 'Hello World!',
        shopping_list: [
            { name: 'bread', price: 2.75 },
            { name: 'milk' , price: 2.50 },
            { name: 'pasta', price: 1.99 }
        ]
    },
    methods: {
        calculateTotal: function(){
            //Your code here
        }
    }
});
```

4. In your HTML, print out the result of ‘calculateTotal’

```
<h2>{{ calculateTotal() }}</h2>
```

5. Open in your browser to test

2.4 Exercise 5: Components

This lab has given a brief introduction to VueJS. Over the next couple of weeks we shall be going over some of the more complex features. However, there is not enough time in the labs to teach everything that Vue has to offer. For more features and help with Vue, consult the Vue documentation: https://vuejs.org/v2/guide

For this exercise, use the Vue documentation (https://vuejs.org/v2/guide/index.html#Composing-with-Components) to rewrite how we list our shopping list items using components.