Module 3-11

Introduction to VUE

Component Based JS

- VUE.js is a component based JS Framework.
- Component based frameworks are very popular now (2020), some other frameworks you might have heard of: React and Angular.
- A component is a reusable piece of code that behaves in a “plug and play” manner, easily incorporated to other parts of the code base.
- Here is some market share data on all the various frameworks: https://www.freecodecamp.org/news/complete-guide-for-front-end-developers-javascript-frameworks-2019/

Anatomy of a VUE Component

```
<template>
   <div class="main">
       <h2>Product Reviews for {{ name }}</h2>

       <p class="description">{{ description }}</p>
   </div>
</template>

<style scoped>
div.main {
 margin: 1rem 0;
}
</style>

<script>
export default {
 name: 'product-review',
 data() {
   return {
     name: 'Widget',
     description: 'Working as intended.'
   }
 }
}
</script>
```

A VUE component is made of up of three parts:

- The **<template>** section which contains HTML code.
- The **<style>** section which contains CSS code.
- The **<script>** section which contains JavaScript code.

What the user sees on the screen is defined mostly by the HTML elements created in the template section and any CSS styles applied in the style section.

The behavior of the component is defined by what's in the script section.

JS Objects in VUE

Within VUE the data function can define a JS object.

```
<script>
export default {
 name: 'product-review',
 data() {
   return {
     name: 'Widget',
     description: 'Working as intended.'
   }
 }
}
</script>
```

Note that we are returning a JS object with 2 properties, a name, and a description.

Displaying Data on the Template

Consider the following code:

```
<script>
export default {
 name: 'product-review',
 data() {
   return {
     name: 'Widget',
     description: 'Working as expected'
   }
 }
}
</script>
```

```
<template>
   <div class="main">
       <h2>Product Reviews for {{ name }}</h2>

       <p class="description">{{ description }}</p>

   </div>
</template>
```

The HTML page will render the following:

Formatting the Template

The <style> section can contain any valid CSS rules. Consider the code below, which applies some formatting to a div with a class name of main (which happens to be the class name on the previous slide):

```
<style scoped>
div.main {
 margin: 1rem 0;
}
</style>
```

```
<template>
   <div class="main">
       <h2>Product Reviews for {{ name }}</h2>

       <p class="description">{{ description }}</p>

   </div>
</template>
```

Integrating All your Components

- The key benefit of using a component based framework is that we can take a bunch of components and bring them together for our final product.

- After we've created all necessary components, we can integrate them into the App.vue file.

- Let's assume that the component we just build is called **ProductReview.vue**. Let's also assume that there is another component called **HelloWorld.vue**.

The App.VUE file:

To integrate the two components, we need to add the following code

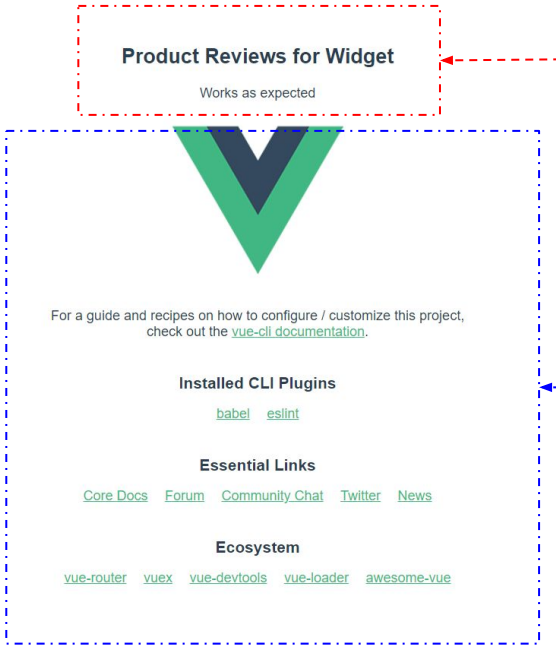
```
<template>
 <div id="app">
   <ProductReview/>
   <img alt="Vue logo" src="./assets/logo.png">
   <HelloWorld/>
 </div>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue';
import ProductReview from './components/ProductReview.vue';

export default {
 name: 'app',
 components: {
   HelloWorld,
   ProductReview
 }
}
</script>
```

Here we have integrated both components into the main VUE app. Note the following checklist:

- In <script> the components have been imported
- In <script>, the components object is populated with the two component names.
- You are rendering the components in the <template>

Integrating All your Components

This is the final result, two fully integrated components:



The *ProductReview* component

The *HelloWorld* component

Let’s look at today’s lecture code

VUE Building Blocks

VUE-Directives

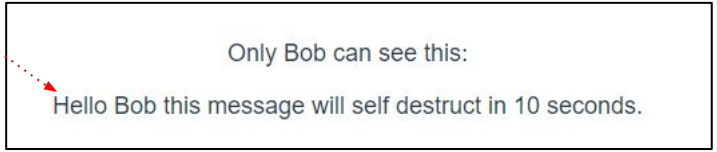
———

Before we get started on data-binding let’s introduce several VUE directives.

- A VUE directive is an extra attribute on a HTML element that asks the VUE library to take some kind of action on that element.
- Today, we will discuss the following:
  - **v-model**: directly associates a DOM element to a chunk of the JSON model.
  - **v-for:** (with v-bind): loops
  - **v-if**: renders the DOM element if certain conditions are met.

v-if

```
<template>
   <div class="main">
       <p>Only Bob can see this:</p>
       <p class="description"  v-if="name == 'Bob'">Hello {{name}} this
        message will self destruct in 10 seconds.</p>
   </div>
</template>


<script>
export default {
 name: 'product-review',
 data() {
   return {
     name: 'Bob',
     description: 'secret agent'
   }
 }
}
</script>
```



The v-if directive will render a DOM element only if certain conditions are met.

Note that the second paragraph has a v-if directive. The element will only display if the name attribute is Bob.

v-for

```
<template>
   <div class="main">
       <p>List of Employees:</p>
       <ul>
         <li v-for=' employee in empList' :key='employee'>{{ employee}} </li>
       </ul>
   </div>
</template>

<script>
export default {
 name: 'product-review',
 data() {
   return {
     empList: ['Alice','Bob','Charlie']
   }
 }
}
</script>
```

List of Employees:

- Alice
- Bob
- Charlie

The v-for directive is used for looping. We want to apply the v-for on the HTML element that is going to repeat.

In here, we are looping through an array of Strings, the <li> element will be repeated three times, one for each element on the array.

v-for : (but with an array of objects)

```
<template>
   <div class="main">
       <p>List of Employees:</p>
       <ul>
<li v-for='employee in empList' :key='employee'>
{{employee.id}} > {{employee.name}}
</li>
       </ul>
   </div>
</template>
<script>
export default {
 name: 'product-review',
 data() {
   return {
     empList: [
       {id: 1, name: 'Alice'},
       {id: 2, name: 'Bob'},
       {id: 3, name: 'Charlie'}
     ]
   }
 }
}
</script>
```

List of Employees:

- 1 > Alice
- 2 > Bob
- 3 > Charlie

In the previous example we had an array of Strings, now we are working with an array of objects, necessitating dot notation, i.e. employee.name

Let’s use the v-for directive

Computed Properties

Computed properties can be thought of as custom fields based on the JSON data model. Computed properties are defined in the script section of a VUE component:

```
<script>
export default {
 name: 'product-review',
 data() {
       ...
 },
 computed: {
   metricUnits() {
      let metricMeasure = this.volumeImperial / 0.061024;
      return metricMeasure;
   }
 }
}
</script>
```

- The way computed properties are defined greatly resemble functions!
- Note that in relation to the data() section, the computed section is a peer (not a descendant) of data.

Computed Properties

We can now refer to these computed properties using the double mustache.

```
<template>
   <div class="main">
       <h2>Product Reviews for {{ name }}</h2>
       <p class="description">{{ description }}</p>
       <p>Volume in Imperial Units: {{  volumeImperial }}</p>
       <p>Volume in Metric Units:{{  metricUnits }}</p>
   </div>
</template>
```

```
<script>
export default {
 name: 'product-review',
 data() {
   return {
     name: 'Cigar Parties for Dummies',
     description: 'Banned in 50 countries',
     volumeImperial: '100'
   }
 },
 computed: {
    metricUnits() {
       let metricMeasure = this.volumeImperial /
0.061024;
       return metricMeasure;
    }
 }
}
</script>
```

Let’s create some computed properties

Data Binding Definition

- Data Binding techniques allow your HTML data-dependent elements to remain synchronized with its data source.
  - Consider the case of a drop-down box on HTML that lists all the Canadian provinces and US states… you could write A LOT of HTML and build this drop-down.
    - Or… you could bind the box to a JSON representation of the data.
- We have already seen plenty of examples for one way data binding where the HTML comment is derived from the JSON object inside the script section.

Data Binding: one way binding

- In VUE, one way binding is used to associate some aspect of a data object with the property of a HTML element.
- The VUE directive **v-bind** is used for this purpose.

V-bind example

```
data() {
    return {
      name: "Cigar Parties for Dummies",
      description:
        "Host and plan the perfect cigar party",
      reviews: [
        {
          reviewer: "Malcolm Gladwell",
          title: "What a book!",
          rating: 3,
          favorited: false
        }
      ]
… }
```

```
<div
  class="review"
  v-bind:class="{favorited: review.favorited}"
  v-for="review in reviews"
  v-bind:key="review.id"
>
```

We are binding the property favorited within the review object to the div's class property.

If the value of favorited is true, the CSS class **favorited** will be applied. The definition of this CSS class will be in the style section.

Data Binding: two way binding

- In VUE two way binding is used to tie the observed value of a HTML element along with the value of a data object property.
- Changing either affects the other.
- The VUE Directive **v-model** is used for this purpose.

V-Model Example

```
data() {
    return {
      name: "Cigar Parties for Dummies",
      description:
        "Host and plan the perfect cigar party",
      reviews: [
        {
          reviewer: "Malcolm Gladwell",
          title: "What a book!",
          rating: 3,
          favorited: false
        }
      ]
… }
```

```
<input type="checkbox" v-model="review.favorited" />
```

The value of the checkbox (either checked or unchecked) will be consistent with the true or false value of the property “favorited.”

Let’s work with some binding examples

Creating a VUE project from scratch

To create a VUE project from scratch called scratchpad, you can enter the following command:

vue create scratchpad