

Module 4-9

VUE building blocks
Data Binding

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

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

```
<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>
```

Only Bob can see this:

Hello Bob this message will self destruct in 10 seconds.

Note that the second paragraph has a v-if directive.

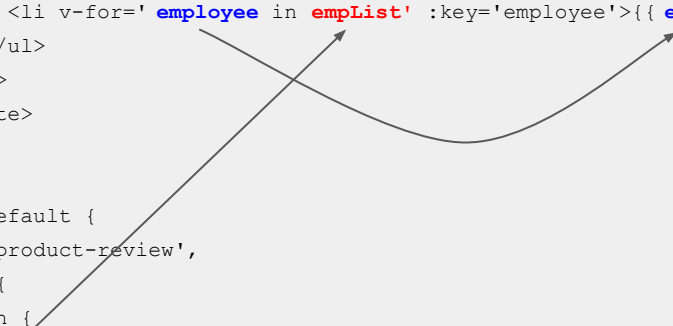
The element will only display if the name attribute is Bob.

v-for

The v-for directive is used for looping. This operates in a similar manner as `<c: foreach>` in JSTL. We want to apply the v-for on the HTML element that is going to repeat!

```
<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

In here, we are looping through an array of Strings, the `` 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

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>
```

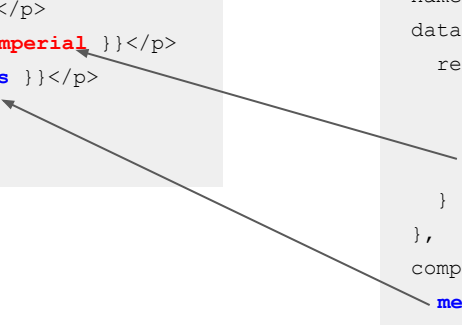
- The way computed properties are defined greatly resemble functions!
- Note that in relation to the data() section, the computed section is a peer (not a descendant) of data.

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>
```



A comprehensive example

Binding

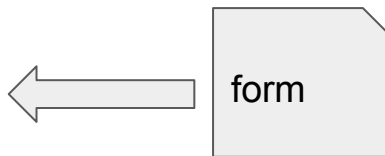
Data Binding Definition

- 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 content is derived from the JSON object inside the script section.

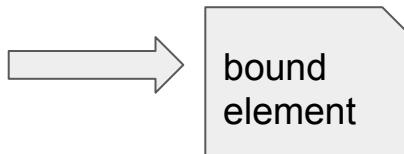
Two way binding: a visual

Suppose we had the following JSON object:

```
data() {  
  return {  
    review: {  
      title: "Hello",  
      reviewer: "",  
      rating: "",  
      review: ""  
    }  
  };  
}
```



In our view, we have a form, input from the form will update the values of the data model.



The current values from the data model will be reflected on a bound element within the view

Two way binding: v-model

Let's take a look at part of the form that will update the data model first using v-model:

```
<template>
  <div class="container">
    <h1>Add New Review</h1>
    <div class="row">
      <div class="col-7">
        <form>
          <div class="form-group">
            <label for="title">Title</label>
            <input
              type="text"
              class="form-control"
              id="title"
              placeholder="Enter title"
              v-model="review.title"
            />
          </div>
        </div>
      </div>
    </div>
  </div>
  ...

```

```
data() {
  return {
    review: {
      title: "Hello",
      reviewer: "",
      rating: "",
      review: ""
    }
  };
}
```

Note how v-model allows us to associate a form element with the JSON data model.

Two way binding: Bound Elements

We can have an element

```
<template>
<div class="container">
  <h1>Add New Review</h1>
  <div class="row">
    <div class="col-7">
      <form>
        <div class="form-group">
          <label for="title">Title</label>
          <input
            type="text"
            class="form-control"
            id="title"
            placeholder="Enter title"
            v-model="review.title"
          />
        </div>
      </form>
    </div>
  </div>
</div>
...
```

```
data() {
  return {
    review: {
      title: "Hello",
      reviewer: "",
      rating: "",
      review: ""
    }
  };
}
```

Note how v-model allows us to associate a form element with the JSON data model.

Two way binding: Bound Elements

We can use a mustache to have an HTML element reflect the value of the data model:

```
data() {  
  return {  
    review: {  
      title: "Hello",  
      reviewer: "",  
      rating: "",  
      review: ""  
    }  
  };  
}
```

The value of the
JSON object will be
properly reflected
on the view.

```
<div class="col-5">  
  <h2>Submission</h2>  
  <hr />  
  <p>Title: {{ review.title }}</p>  
  <p>Reviewer: {{ review.reviewer }}</p>  
  <p>Rating: {{ review.rating }}</p>  
  <p>Review: {{ review.review }}</p>  
</div>
```

An example of two way binding