



School of Computer Science

Web and Database Computing 2019

Lecture 19: Vue Conditions, Events & Components

adelaide.edu.au

seek LIGHT

Complex tasks with simple templates

You can follow along in the lecture slides,
but also following the guide at <https://vuejs.org/v2/guide/>

Conditional Rendering

- We can use Vue template syntax to set content and attributes, but what if we wanted content to change large blocks of html?
- One way we can do this is Conditional Rendering:

Vue HTML Result

[Edit in JSFiddle](#)

```
var vueApp = new Vue({  
  el: "#app",  
  data: {  
    property1: true,  
    property2: 'Hi'  
  }  
})
```

https://jsfiddle.net/ian_knight_uofa/vmsxh4rj/7/

Conditional Rendering; What's happening?

- Element is only rendered if the condition is true.

```
<h1 v-if="condition">  
  This heading only rendered if  
  data property 'condition' evaluates true.  
</h1>
```

- Can also have **else if** and **else** conditions.

```
<h1 v-else-if="condition">Hello</h1>  
<h1 v-else>Hi</h1>
```

- Elements must be immediate siblings for this to work (no other elements between).

Vue HTML Result [Edit in JSFiddle](#)

```
var vueApp = new Vue({  
  el: "#app",  
  data: {  
    property1: true,  
    property2: 'Hi'  
  }  
})
```

Conditional Rendering; What's happening?

- Conditions can also be expressions.

```
<h1 v-if="property === 'value'">  
  This heading only rendered if data property  
  'property' is strictly equaly to 'value'.  
</h1>
```

Vue HTML Result

[Edit in JSFiddle](#)

```
var vueApp = new Vue({  
  el: "#app",  
  data: {  
    property1: true,  
    property2: 'Hi'  
  }  
})
```

Repetition for Arrays

- We often have a list of objects or data that we want rendered into a set of items.
- Vue provides list rendering for this:

Vue HTML Result

[Edit in JSFiddle](#)

```
var vueApp = new Vue({  
  el: "#app",  
  data: {  
    headingList: ['A heading',  
                  'Another heading']  
  }  
});
```

https://jsfiddle.net/ian_knight_uofa/2hyerdwv/6/

List Rendering; What's happening?

- Data properties that are Arrays can be iterated over using foreach style syntax.

```
<div id="vueHeadings">  
  <h1 v-for="item in list">  
    Creates a set of similar elements  
    where {{ item }} will be replaced with  
    each of the values in the data property  
    'list' which is an array.  
  </h1>  
</div>
```

```
new Vue({  el: "#vueHeadings",  
          data: {  
            list: ['value1', 'value2']  
          }  
});
```

HTML

Vue

Result

[Edit in JSFiddle](#)

```
<div id="app">  
  <h1 v-for="item in headingList">{{ item }}</h1>  
</div>  
<button onclick="vueApp.headingList.push('Yet another heading')">Add heading</button>
```

List Rendering; What's happening?

- We can also include the index of list items when iterating.

```
<ul id="vueList">  
  <li v-for="(item, index) in list">  
    Creates a set of similar elements  
    where {{ index }} will be replaced with  
    the array index of each {{ item }} in  
    the data property 'list' (array).  
  </li>  
</ul>
```

```
new Vue({  el: "#vueList",  
          data: {  
            list: ['value1', 'value2']  
          }});
```

Vue HTML Result [Edit in JSFiddle](#)

```
var vueApp = new Vue({  
  el: "#app",  
  data: {  
    headingList: ['A heading',  
                  'Another heading']  
  }  
});
```


List Rendering; What's happening?

- We can also iterate over objects.

```
<ul id="vueList">
  <li v-for="(value, key) in object">
    Creates a set of similar elements where
    {{ key }} will be replaced with the name
    of an objects's property and {{ value }}
    contains the value for that key.
  </li>
</ul>
```

```
new Vue({ el: "#vueList",
  data: {
    obj1 : { prop1: 'value1',
              prop2: 2 }
  }});
```

Vue HTML Result [Edit in JSFiddle](#)

```
var vueApp = new Vue({
  el: "#app",
  data: {
    headingList: ['A heading',
                  'Another heading']
  }
});
```

Reacting to Events

- Writing event handlers can be complex when multiple items are to use the same handler.
- Vue provides its own way of declaring event handlers that ensures the event is linked with the correct Vue instance:

HTML Vue Result

[Edit in JSFiddle](#)

```
<div id="example">
  <button v-on:click="greet">{{ name }}></button>
</div>
```

https://jsfiddle.net/ian_knight_uofa/194sjx6o/4/

Event Handling; What's happening?

- We can connect events to our Vue instance.

```
<button v-on:click="greet">
```

```
new Vue({
  el: '#example',
  data: {
    name: 'Hello'
  },
  // define methods under the `methods` object
  methods: {
    greet: function (event) {
      // `this` points to the Vue instance
      this.name = "Hi";
    }
  }
});
```

Vue HTML Result [Edit in JSFiddle](#)

```
new Vue({
  el: '#example',
  data: {
    name: 'Hello'
  },
  // define methods under the `methods` object
  methods: {
    greet: function (event) {
      // `this` inside methods points to the Vue instance
      this.name = "Hi";
    }
  }
});
```

Forms

- We often want to retrieve data from forms.
- Vue provides a way easily accessing and working with form elements using **v-model**:

HTML Vue Result

[Edit in JSFiddle](#)

```
<div id="app">
  <input type="text" v-model="message" placeholder="edit me" />
<p>Message is: {{ message }}</p>
</div>
```

https://jsfiddle.net/ian_knight_uofa/wxzc53y6/4/

Form Bindings; What's happening?

- We also easily access/set the values of input fields.

```
<input type="text" v-model="message" />
```

```
new Vue({  
  el: "#app",  
  data: {  
    message: 'text'  
  }  
});
```

- Where the data property message contains the value of the text box.
 - Updating message will update the checkbox.
 - Note: the initial value comes from Vue, not what's set in the HTML.

Vue HTML Result

[Edit in JSFiddle](#)

```
new Vue({  
  el: "#app",  
  data: {  
    message: 'text'  
  }  
});
```

Form Bindings; What's happening?

- We also easily access multiple checkboxes/radio buttons.

```
<input type="checkbox" value="Jack" v-model="checkedNames"/>  
<input type="checkbox" value="John" v-model="checkedNames"/>  
<input type="checkbox" value="Mike" v-model="checkedNames"/>
```

```
new Vue({  
  el: "#app",  
  data: {  
    checkedNames: []  
  }  
});
```

- Where the data property checkedNames is an array that contains the values of all checked items.

Vue HTML Result [Edit in JSFiddle](#)

```
new Vue({  
  el: "#app",  
  data: {  
    checkedNames: []  
  }  
});
```

Bringing it all together

Using components for Object Oriented Web Design

Components

So far our HTML and JavaScript have been kept very separate. Vue allows us to combine them to create reusable interactive components.

Vue HTML Result

[Edit in JSFiddle](#)

```
// Define a new component called blog-post
Vue.component('blog-post', {
  props: ['content'],
  data: function () {
    return {
      likes: 0
    }
  },
  template: '<div class="blogPost"> \
<p>{{ content }}</p> \
<button v-on:click="likes++">Like</button> \
<span>{{ likes }}</span> \
</div>'
})
```

https://jsfiddle.net/ian_knight_uofa/x2vdo3c1/22/

Components; What's happening?

- Components allow us to define reusable groups of elements and template data.

```
Vue.component('blog-post', {
  data: function () {
    return {
      likes: 0
    }
  },
  template: '<div class="blogPost">'+
    '<button v-on:click="likes++;">Like</button>'+
    '<span>{{ likes }}</span>'+
    '</div>'
});
```

- Where **template** contains the HTML for the component.

Vue HTML Result [Edit in JSFiddle](#)

```
// Define a new component called blog-post
Vue.component('blog-post', {
  props: ['content'],
  data: function () {
    return {
      likes: 0
    }
  },
  template: '<div class="blogPost"> \
<p>{{ content }}</p> \
<button v-on:click="likes++;">Like</button> \
<span>{{ likes }}</span> \
</div>'
});

new Vue({
  el: "#app",
  data: {
    currentpost: 'Write your post here',
    posts: [
      { text: "A blog post" },
      { text: "A different post" }
    ]
  },
  methods: {
```

Components; What's happening?

- We can use a component by creating a matching tag in our HTML.

```
<div id="app">
  <blog-post></blog-post>
</div>
```

- Needs to be done inside a Vue instance (#app in this example).

Vue HTML Result [Edit in JSFiddle](#)

```
// Define a new component called blog-post
Vue.component('blog-post', {
  props: ['content'],
  data: function () {
    return {
      likes: 0
    }
  },
  template: '<div class="blogPost"> \
<p>{{ content }}</p> \
<button v-on:click="likes++">Like</button> \
<span>{{ likes }}</span> \
</div>'
});

new Vue({
  el: "#app",
  data: {
    currentpost: 'Write your post here',
    posts: [
      { text: "A blog post" },
      { text: "A different post" }
    ]
  },
  methods: {
```

Components; What's happening?

- Components can be combined with list-rendering, attributes, and event handlers.

```
<div id="app">
  <blog-post v-bind:content="post.text"
             v-for="post in posts"></blog-post>
</div>
```

```
new Vue({
  el: "#app",
  data: {
    posts: [
      { text: "A blog post" },
      { text: "A different post" }
    ]
  }
});
```

Vue HTML Result Edit in JSFiddle

```
// Define a new component called blog-post
Vue.component('blog-post', {
  props: ['content'],
  data: function () {
    return {
      likes: 0
    }
  },
  template: '<div class="blogPost"> \
<p>{{ content }}</p> \
<button v-on:click="likes++">Like</button> \
<span>{{ likes }}</span> \
</div>'
});

new Vue({
  el: "#app",
  data: {
    currentpost: 'Write your post here',
    posts: [
      { text: "A blog post" },
      { text: "A different post" }
    ]
  },
  methods: {
```

Components; What's happening?

- Data can be passed into components using bound parameters.

```
<div id="app">
  <blog-post v-bind:content="post"
             v-for="post in posts"></blog-post>
</div>
```

```
new Vue({
  el: "#app",
  data: {
    posts: [ "A blog post", "Another post" ]
  }
});
Vue.component('blog-post', {
  props: ['content'],
  template: '<p>{{ content }}</p>'
});
```

Vue HTML Result Edit in JSFiddle

```
// Define a new component called blog-post
Vue.component('blog-post', {
  props: ['content'],
  data: function () {
    return {
      likes: 0
    }
  },
  template: '<div class="blogPost"> \
    <p>{{ content }}</p> \
    <button v-on:click="likes++">Like</button> \
    <span>{{ likes }}</span> \
  </div>'
});

new Vue({
  el: "#app",
  data: {
    currentpost: 'Write your post here',
    posts: [
      { text: "A blog post" },
      { text: "A different post" }
    ]
  },
  methods: {
```

Components; What's happening?

- Components can have their own data and event handlers.

```
Vue.component('blog-post', {
  props: ['content'],
  data: function () {
    return { likes: 0 };
  },
  methods: {
    addLike: function(){
      this.likes++;
    }
  },
  template: '<div><p>{{ content }}</p>'+
    '<button v-on:click="addLike">Like</button>'+
    '<span>{{ likes }}</span></div>'
});
```

Vue HTML Result [Edit in JSFiddle](#)

```
// Define a new component called blog-post
Vue.component('blog-post', {
  props: ['content'],
  data: function () {
    return {
      likes: 0
    }
  },
  template: '<div class="blogPost"> \
<p>{{ content }}</p> \
<button v-on:click="likes++">Like</button> \
<span>{{ likes }}</span> \
</div>'
});

new Vue({
  el: "#app",
  data: {
    currentpost: 'Write your post here',
    posts: [
      { text: "A blog post" },
      { text: "A different post" }
    ]
  },
  methods: {
```

- Note **data** is a function that returns the data variables; this is due to JavaScript variable scope.



THE UNIVERSITY
of ADELAIDE



What's happening

Due:

- Prac Exercise 5 now available, Fri week 7.
- Prac Exercise 6 coming soon.

Next week:

- Have a great break!

Further learning:

- Start on your group project