Styles and Classes

In this section, we will discuss styles and classes and the different ways to apply/manage styles in a Vue project.

Common Style Patterns

1. **Global Styles**: Importing styles via your `index.html` file using the `link` tag works as expected:

```
HTML
<link rel="stylesheet" src="./styles.css" />
```

In as much as Vue's building blocks are components, you could decide to not have specific styles per component. In such case, Global styles are your best option.

1. **Component Styles**: Vue's Single File Components allows you to define your styles for each component. When doing so, you can scope the styles to its respective component. This will help avoid a situation where the style leaks and affects elements that are not related to the component.

Example:

Table of Contents


```
HTML
<template></template>
<script></script>

<style scoped>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

1. **Inline Styles**: Inline styles will also work. All the considerations you know about inline styles still apply to a Vue app. Therefore, care should be taken when using this pattern:

```
HTML
<template>
  <h1 style="font-size: 36px">Title</h1>
</template>
```

Inline styles are also used to apply dynamic styles to HTML elements in Vue. Let's discuss how styles and classes are added dynamically.

Dynamic Styles and Classes

Styling is not a way one street. Most times, you might not build an app that only depends on static styles and classes. It is very normal to have a reason to update styles and classes on the fly, using JavaScript.

Vue simplifies this task with style/class bindings; making dynamic styles and classes a sweet story to tell.

Dynamic Styles

Styles are mostly applied to elements using JavaScript objects:

```
HTML
<div v-bind:style="{ color: 'red', backgroundColor: 'blue' }"></div>
```

Style bindings are achieved with the `v-bind` directive. It takes a JS object with the properties represented as JS object keys and values represented as JS object values.

Vue allows you to use camel cased version of the css property or the usual kebab case. When using the kebab case, you must wrap the property in quotes:

```
HTML
<div v-bind:style="{ color: 'red', 'background-color': 'blue' }"></div>
```

Where this becomes interesting is, you can provide the style values via your Vue logic:

```
HTML
<div v-bind:style="{ color: 'red', backgroundColor: bgColor }"></div>

<script>
  data () {
    return {
      bgColor: 'rgb(0, 0, 255)'
    }
  }
</script>
```

Better still:

```
HTML
<div v-bind:style="myStyle"></div>
<script>
  data () {
    return {
      myStyle: { color: 'red', backgroundColor: 'rgb(0,0,255)' }
    }
  }
</script>
```

You can also bind multiple styles using the array option:

```
HTML
<div v-bind:style="[myStyle, ourStyle]"></div>
<script>
  data () {
    return {
      myStyle: { color: 'red', backgroundColor: 'rgb(0,0,255)' },
      ourStyle: { fontSize: '32px', position: 'relative' },
    }
  }
</script>
```

Dynamic Classes

You can toggle whether a class should be added to an element or not. This is powerful when you want to group set of styles in a class outside your logic; probably in a stylesheet.

```
HTML
<div v-bind:class="{ active: isAuth }">
...
</div>

<script>
  data () {
    return {
      isAuth: true
    }
  }
</script>
```

Dynamic classes are applied using the `v-bind` directive. In the example shown above, the `active` class will be applied if `isAuth` resolves to be truthy.

You can apply multiple classes with multiple conditions:

```
HTML
<div v-bind:class="{ active: isAuth, 'admin-flag': isAdmin }">
...
</div>

<script>
  data () {
    return {
      isAuth: true,
      isAdmin: false
    }
  }
</script>
```

It's worth noting that classes applied to a component selector will be passed down to the component's root element. For example, if we have the following component:

```
28
Vue.component('product', {
  template: '<div class="card">...</div>'
})
```

When used as such:

```
HTML
<product class="{selected: isSelected}"></product>
```

If `isActive` is true, the following will be rendered:

```
HTML
<div class="card selected">...</div>
```