How to Dynamically Add a Class Name in Vue

Being able to add a dynamic class name to your component is really powerful.

It lets you write custom themes more easily, add classes based on the state of the component, and also write different variations of a component that rely on styling.

Adding a dynamic class name is as simple as adding the prop `:class="classname"` to your component. Whatever `classname` evaluates to will be the class name that is added to your component.

Of course, there is *a lot* more we can do here with dynamic classes in Vue.

We'll cover a lot of stuff in this article:

- Using static and dynamic classes in Vue
- How we can use regular Javascript expressions to calculate our class
- The array syntax for dynamic class names
- The object syntax (for more variety!)
- Generating class names on the fly
- How to use dynamic class names on custom components

Not only that, but at the end I'll show you a simple pattern for using computed props to help clean up your templates. This will come in handy

once you start using dynamic class names everywhere!

Let's dive in!

Static and Dynamic Classes

In Vue we can add both static and dynamic classes to our components.

Static classes are the boring ones that never change, and will always be present on your component. On the other hand, **dynamic** classes are the ones we can add and remove things change in our application.

If we wanted to add a static class, it's exactly the same as doing it in regular HTML:

```
<template>
  <span class="description">
    This is how you add static classes in Vue.
  </span>
</template>
```

Dynamic classes are very similar, but we have to use Vue's special property syntax, `v-bind`:

```
<template>
  <span v-bind:class="'description'">
    This is how you add static classes in Vue.
  </span>
</template>
```

You'll notice we had to add extra quotes around our dynamic class name.

This is because the `v-bind` syntax takes in whatever we pass as a Javascript value. Adding the quotes makes sure that Vue will treat it as a string.

Vue also has a shorthand syntax for `v-bind`:

```
<template>
  <span :class="'description'">
    This is how you add static classes in Vue.
  </span>
</template>
```

What's really neat is that you can even have both static and dynamic classes on the same component.

This let's you have some static classes for the things you know won't change, like positioning and layout, and dynamic classes for your theme:

```
<template>
  <span
    class="description"
    :class="theme"
  >
    This is how you add static classes in Vue.
  </span>
</template>
```

```
export default {
  data() {
    return {
      theme: 'blue-theme',
    };
  }
};
```

```
.blue-theme {
  color: navy;
  background: white;
}
```

In this case, `theme` is the variable that contains the classname we will apply (remember, it's treating it as Javascript there).

Using a Javascript Expression

Since `v-bind` will accept any Javascript expression, we can do some pretty cool things with it.

Guard Expressions

There is a cool trick using the logical `&&` that allows us to conditionally apply a class:

```
<template>
  <span
    class="description"
    :class="useTheme && theme"
  >
    This is how you add dynamic classes in Vue.
  </span>
</template>
```

This is known as a guard expression.

But how does it work?

Here we have the variable `useTheme` which is a boolean, and `theme` is the value of the theme class.

In Javascript, the AND operator will short-circuit if the first value is `false`.

Since both values need to be `true` in order for the expression to be `true`, if the first is `false` there is no point in checking what the second one is, since we already know the expression evaluates to `false`.

So if `useTheme` is `false`, the expression evaluates to `false` and no dynamic class name is applied.

However, if `useTheme` is true, it will also evaluate `theme`, and the expression will evaluate to the value of `theme`. This will then apply the value of `theme` as a classname.

The Many Ways I Use Ternaries

We can do a similar trick with ternaries.

If you aren't familiar, a ternary is basically a short-hand for an if-else statement.

They look like this:

```
const result = expression ? ifTrue : ifFalse;
```

Sometimes though, we'll format them like this for readability:

```
const result = expression
  ? ifTrue
  : ifFalse;
```

If `expression` evaluates to `true`, we get `ifTrue`. Otherwise we will get `ifFalse`.

Their main benefit is that they are concise, and count as only a single statement. This lets us use them inside of our templates.

Ternaries are useful if we want to decide between two different values:

```
<template>
  <span
    class="description"
    :class="darkMode ? 'dark-theme' : 'light-theme'"
  >
    This is how you add dynamic classes in Vue.
  </span>
</template>
```

If `darkMode` is `true`, we apply `dark-theme` as our class name. Otherwise we choose `light-theme`.

Now let's figure out how to add multiple dynamic class names at the same time!

Using the Array Syntax

If there are lots of different classes you want to add dynamically, you can use arrays or objects. Both are useful, but we'll cover arrays first.

Since we are just evaluating a javascript expression, you can combine the expressions we just learned with the array syntax:

```
<template>
  <span
    class="description"
    :class="[
      fontTheme,
      darkMode ? 'dark-theme' : 'light-theme',
    ]"
  >
    This is how you add dynamic classes in Vue.
  </span>
</template>
```

What's going on here?

We are using the array to set two dynamic class names on this element.

The value of `fontTheme` is a classname that will change how our fonts look.

From our previous example, we can still switch between light and dark themes using our `darkMode` variable.

Using the Object Syntax

We can even use an object to define the list of dynamic classes, which gives us some more flexibility.

For any key/value pair where the value is `true`, it will apply the key as the classname.

Let’s look at an example of the object syntax:

```
<template>
  <span
    class="description"
    :class="{
      'dark-theme': darkMode,
      'light-theme': !darkMode,
    ]"
  >
    This is how you add dynamic classes in Vue.
  </span>
</template>
```

Our object contains two keys, `dark-theme` and `light-theme`. Similar to the logic we implemented before, we want to switch between these themes based on the value of `darkMode`.

When `darkMode` is `true`, it will apply `dark-theme` as a dynamic class name to our element. But `light-theme` won’t be applied because `!darkMode` will evaluate to `false`.

The opposite happens when `darkMode` is set to false. We get `light-theme` as our dynamic classname instead of `dark-theme`.

> *It is common convention to use dashes — or hyphens — in CSS classnames. But in order to write the object keys with dashes in Javascript, we need to surround it in quotes to make it a string.*

Now we've covered the basics of dynamically adding classes to Vue components.

How do we do this with our own custom components?

Using with Custom Components

Let's say that we have a custom component that we are using in our app:

```
<template>
  <MovieList
    :movies="movies"
    :genre="genre"
  />
</template>
```

If we want to dynamically add a class that will change the theme, what would we do?

It's actually really simple.

We just add the `:class` property like before!

```
<template>
  <MovieList
    :movies="movies"
    :genre="genre"
    :class="darkMode ? 'dark-theme' : 'light-theme'"
  />
</template>
```

The reason this works is that Vue will set `class` on the root element of `MovieList` directly.

When you set props on a component, Vue will compare those props to what the component has specified in it's `props` section. If there is a match, it will pass it along as a normal prop. Otherwise, Vue will add it to the root DOM element.

Here, because `MovieList` didn't specify a `class` property, Vue knows that it should set it on the root element.

There are some even more advanced things we can do with dynamic class names though...

Generating Class Names on the Fly

We've seen a lot of different ways that we can dynamically *add* or *remove* class names.

But what about dynamically *generating* the class name itself?

Let's say you have a `Button` component, with 20 different CSS styles for all of your different types of buttons.

That's a lot of variation!

You probably don't want to spend your whole day writing out every single one, along with the logic to turn it on and off. This would also be a nasty mess when it comes time to update the list!

Instead, we'll **dynamically generate the name of the class** we want to apply.

A simple version of this you've already seen:

```
<template>
  <span
    class="description"
    :class="theme"
  >
    This is how you add static classes in Vue.
  </span>
</template>
```

```
export default {
  data() {
    return {
      theme: 'blue-theme',
    };
  }
};
```

```
.blue-theme {
  color: navy;
  background: white;
}
```

We can set a variable to contain the string of whatever class name we want.

If we wanted to do this for our `Button` component, we could do something simple like this:

```
<template>
  <button
    @click="$emit('click')"
    class="button"
    :class="theme"
  >
    {{ text }}
  </button>
</template>
```

```
export default {
  props: {
    theme: {
      type: String,
      default: 'default',
    }
  }
};
```

```
.default {}

.primary {}

.danger {}
```

Now, whoever is using the `Button` component can simply set the `theme` prop to whatever theme they want to use.

If they don't set any, it will add the `.default` class.

If they set it to `primary`, it will add the `.primary` class.

Cleaning Things Up With Computed Props

Eventually the expressions in our template will get too complicated, and it will start to get very messy and hard to understand.

Luckily, we have an easy solution for that.

If we convert our expressions to computed props, we can move more of the logic *out of the template* and clean it up:

```
<template>
  <MovieList
    :movies="movies"
    :genre="genre"
    :class="class"
  />
</template>
```

```
export default {
  computed: {
    class() {
      return darkMode ? 'dark-theme' : 'light-theme';
    }
  }
};
```

Not only is this much easier to read, but it's also easier to add in new functionality and refactor in the future.

This pattern — moving things from the template into computed props — works with all sorts of things as well. It's one of the best tricks for cleaning up your Vue components!