# Template Syntax

NEXT: CONDITIONAL & LIST RENDERING

# Template Syntax

With Vue being an MVVM tool, there needs be a way to bind the models to their respective views. This way, data being handled by an underlying model is printed in the browser for the user's consumption.

Views are represented by HTML templates but with extra utility features like interpolation ( `{{ }}` ) and directives ( `v-on:click` ) to help display the models' data.

# A Comparison

A quick look at the conventional JavaScript DOM manipulation (without Vue) techniques will make us appreciate what Vue's templates have to offer. A simple and perfect example is trying to insert text into a given div:

## Table of Contents



JS

```
// jQuery
const divElement = $('#text');
divElement.text('Hello Vue')
```

It's even more tedious with JavaScript:

JS

```
// JavaScript
const divElement = document.getElementById('text');
const textNode = document.createTextNode('Hello Vue');

divElement.appendChild(textNode);
```

With Vue, it's dead simple and declarative:

JS

```
new Vue({
    template: `<div>{{text}}</div>`,
    data () {
        return {
            text: 'Hello Vue'
        };
    }
}).$mount('#app')
```

The template (view) and data (model) are split into different entities but bound together using the object properties. Let's explain some binding concept:

## Interpolation

The concept of using double curly braces as a placeholder for data properties in a Vue template is known as an interpolation:

HTML

```
<div>{{text}}</div>
```

The double curly braces are popularly known as Mustache syntax/tag. Vue is not the only tool that uses it so don't get thrown off when you see it somewhere else.

The mustache tag will be replaced with what ever the data holds during render. The amazing thing is that when the bound data changes, the template automatically gets updated. This is one painful limitation our JavaScript example has. We would have to check for changes and do the update manually.

## Directives

Directives are custom HTML attributes that allow you to bind data to both view and attributes. It also helps with binding DOM events.

HTML

```
<div v-text="text"></div>

<!-- same as -->
<div>{{text}}</div>
```

## Directive Attributes

You might be tempted to use interpolation (mustache) in HTML attributes:

HTML

```
<!-- WRONG!!! -->
<div id="{{id}}">{{text}}</div>
```

Vue will scream errors once it encounters that.

The `v-bind` directive is responsible for such tasks. So you could safely do this:

HTML

```
<!-- VERY CORRECT!!! -->
<div v-bind:id="id">{{text}}</div>
```

If that seems like a lot of key strokes, shorthand is allowed:

HTML

```
<!-- Vue is awesome !!! -->
<div :id="id">{{text}}</div>

<!-- Another example with href -->
<a :href="url">{{text}}</a>
```

Same pattern works for event binding as well. Rather, than `v-bind` , we will use `v-on` which is Vue's event binding directive:

HTML

```
<button v-on:click="addToCart">Add to Cart</button>

<!-- Shorthand -->
<button @click="addToCart">Add to Cart</button>
```

## Modifiers

Vue allows you to modify the (default) behavior of some attributes right inside the template. A good guess is that you have seen something like this:

JS

```
function handleFormSubmit(event) {
    event.preventDefault();
    // Send to server
}
```

The snippet will make your browser not reload in an attempt to submit a form. You don't have to take this dirty job to the logic again; you can easily use a modifier to handle such case in your template:

HTML

```
<form v-on:submit.prevent="handleFormSubmit">
</form>

<!-- Shorthand -->
<form @submit.prevent="handleFormSubmit">
</form>
```

## Template Expressions

Vue allows you to perform minor JavaScript-like template expressions in the HTML:

HTML

```
<div>{{2 + 5}}</div> <!-- 7 -->

<div>{{2 + '5'}}</div> <!-- 25 -->
```