The Progressive
JavaScript Framework

GET STARTED GITHUB

Installation

npm

```
# latest stable
$ npm install vue
```

bower

```
# latest stable
$ bower install vue
```

Introduction

Hello vue

```
<div id="app">
  {{ message }}
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

The Vue Instance

Structure

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  },
  watch: {
    // whenever question changes, this function will run
    question: function (newQuestion) {
      this.getAnswer()
    }
  },
  computed: {
    reversedMessage: function () {
      return this.message
    }
  },
  methods: {
    my-method: function (newQuestion) {
      this.getAnswer()
    }
  },

    // Lifecycle hooks (beforeCreate,created,beforeMount,mounted,
       beforeUpdate,updated,beforeDestroy,destryed)
  created: function () {
    console.log('created')
  }

})
```

Template Syntax

Text

```
<span>Message: {{ msg }}</span>
<span v-once>This will never change: {{ msg }}</span>
```

```
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}
<div v-bind:id="'list-' + id"></div>
```

Raw HTML

```
<div v-html="rawHtml"></div>
```

Attribute Handling

```
<div v-bind:id="dynamicId"></div>
```

Attribute Modifiers

```
<button v-bind:click.stop="doThis"></button>
```

Filters

```
{{ message | capitalize }}
{{ message | filterA | filterB }}
{{ message | filterA('arg1', arg2) }}
```

Custom filters

```
new Vue({
  // ...
  filters: {
    capitalize: function (value) {
      if (!value) return ''
      value = value.toString()
      return value.charAt(0).toUpperCase() + value.slice(1)
    }
  }
})
```

Directives

Simple directive
```
v-if="seen"
```

with arguments
```
v-bind:href="url"
```

with arguments
```
v-on:click="doSomething"
```

with Modifiers
```
v-on:submit.prevent="onSubmit"
```

Shortcuts

`v-bind` Shorthand

```
<!-- full syntax -->
<a v-bind:href="url"></a>
<!-- shorthand -->
<a :href="url"></a>
```

`v-on` Shorthand

```
<!-- full syntax -->
<a v-on:click="doSomething"></a>
<!-- shorthand -->
<a @click="doSomething"></a>
```

Computed Properties

Basic Example

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // a computed getter
    reversedMessage: function () {
      // `this` points to the vm instance
      return this.message.split('').reverse().join('')
    }
  }
})
```

Watchers

Basic Example

```
<div id="watch-example">
  <p>
    Ask a yes/no question:
    <input v-model="question">
  </p>
  <p>{{ answer }}</p>
</div>
```

```
var watchExampleVM = new Vue({
  el: '#watch-example',
  data: {
      //...
  },
  watch: {
    // whenever question changes, this function will run
    question: function (newQuestion) {
      this.getAnswer()
    }
  },
  methods: {
  }
})
```

Class Bindings

Basic Example

```
<div class="static"
     v-bind:class="{ active: isActive, 'text-danger': hasError }">
</div>
//----------------------
data: {
  isActive: true,
  hasError: false
}
```

Passing an object with classes

```
<div v-bind:class="classObject"></div>
data: {
  classObject: {
    active: true,
    'text-danger': false
  }
}
```

Using computed values for classes

```
<div v-bind:class="classObject"></div>
data: {
  isActive: true,
  error: null
},
computed: {
  classObject: function () {
    return {
      active: this.isActive && !this.error,
      'text-danger': this.error && this.error.type === 'fatal',
    }
```

```
  }
}
```

Using arrays for classes

```
<div v-bind:class="[activeClass, errorClass]">
data: {
  activeClass: 'active',
  errorClass: 'text-danger'
}
```

Conditional classes

```
<div v-bind:class="[isActive ? activeClass : '', errorClass]">
```

Style Bindings

Basic Example

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
data: {
  activeColor: 'red',
  fontSize: 30
}
```

Passing an object for styles

```
<div v-bind:style="styleObject"></div>
data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}
```

Using arrays for styles

```
//multiple style objects
<div v-bind:style="[baseStyles, overridingStyles]">
```

Conditional rendering

If else

```
<div v-if="Math.random() > 0.5">
  Now you see me
</div>
<div v-else>
  Now you don't
</div>
```

v-show

```
<h1 v-show="ok">Hello!</h1>
```

List Rendering

v-for

```
<ul id="example-1">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
```

v-for with index

```
<ul id="example-2">
  <li v-for="(item, index) in items">
    {{ parentMessage }} - {{ index }} - {{ item.message }}
  </li>
</ul>
```

‘of’ instead of ‘in’

```
<div v-for="item of items"></div>
```

Using templates with ‘for’

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider"></li>
  </template>
</ul>
```

Iterating through an object

```
<ul id="repeat-object" class="demo">
  <li v-for="value in object">
    {{ value }}
  </li>
</ul>
```

```
object: {
      FirstName: 'John',
      LastName: 'Doe',
      Age: 30
    }
```

Iterating through an object

```
<div>
  <span v-for="n in 10">{{ n }}</span>
</div>
```

Repeating components

```
<my-component v-for="item in items"></my-component>
```

Repeating components and passing values into the component

```
<my-component
  v-for="(item, index) in items"
  v-bind:item="item"
  v-bind:index="index"> //passing value into
</my-component>
```

Array operations that trigger view updates

- push()
- pop()
- shift()
- unshift()
- splice()
- sort()
- reverse()

Sort (or filter) and display array

```
<li v-for="n in evenNumbers">{{ n }}</li>
```

```
data: {
  numbers: [ 1, 2, 3, 4, 5 ]
},
computed: {
  evenNumbers: function () {
    return this.numbers.filter(function (number) {
      return number % 2 === 0
    })
  }
}
```

Event Handling

Basic

```
<button v-on:click="counter += 1">Add 1</button>
```

Getting event object

```
methods: {
  greet: function (event) {
    // `event` is the native DOM event
  }
}
```

Passing 'event' object with inline statement

```
<button v-on:click="add(a,b, $event)">Add values</button>
```

Using event modifiers

- `.stop - stop propagation`

```
<!-- the click event's propagation will be stopped -->
<a v-on:click.stop="doThis"></a>
```

- `.prevent - prevent default`

```
<!-- just the modifier -->
<form v-on:submit.prevent></form>
```

- `.capture -`

```
<!-- use capture mode when adding the event listener -->
<div v-on:click.capture="doThis">...</div>
```

- `.self`

```
<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div v-on:click.self="doThat">...</div>
```

Keyboard events & modifiers

```
<!-- key modifier using keyAlias -->
<input @keyup.enter="onEnter">
<!-- key modifier using keyCode -->
<input @keyup.13="onEnter">
```

Here's the full list of key modifier aliases:

- enter
- tab
- delete (captures both "Delete" and "Backspace" keys)
- esc
- space
- up
- down
- left
- right

You can also **define custom key modifier aliases** via the global `config.keyCodes` object:

```
// enable v-on:keyup.f1
Vue.config.keyCodes.f1 = 112
```

Form Input Bindings

Text input

```
<input v-model="message" placeholder="edit me">
```

Textarea

```
<textarea v-model="message" placeholder="add multiple lines"></textarea>
```

Single checkbox

```
<input type="checkbox" id="checkbox" v-model="checked">
```

Multiple checkbox

```
<input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
<label for="jack">Jack</label>
<input type="checkbox" id="john" value="John" v-model="checkedNames">
<label for="john">John</label>
<input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
<label for="mike">Mike</label>
```

```
data: {
    checkedNames: []
  }
```

Radio buttons

```
<input type="radio" id="one" value="One" v-model="picked">
<label for="one">One</label>
<br>
<input type="radio" id="two" value="Two" v-model="picked">
<label for="two">Two</label>
```

Select box

```
<select v-model="selected">
  <option>A</option>
```

```
  <option>B</option>
  <option>C</option>
</select>
```

Selectbox with dynamic options

```
<select v-model="selected">
  <option v-for="option in options" v-bind:value="option.value">
    {{ option.text }}
  </option>
</select>
```

Binding to a dynamic variable

```
###CHECKBOX
<input
  type="checkbox"
  v-model="toggle"
  v-bind:true-value="a"
  v-bind:false-value="b">
// when checked:
vm.toggle === vm.a
// when unchecked:
vm.toggle === vm.b
```

```
###RADIO BUTTON
<input type="radio" v-model="pick" v-bind:value="a">
// when checked:
vm.pick === vm.a
```

```
###SELECTBOX
<select v-model="selected">
  <!-- inline object literal -->
  <option v-bind:value="{ number: 123 }">123</option>
</select>
```

```
// when selected:
typeof vm.selected // -> 'object'
vm.selected.number // -> 123
```

Input Modifiers

`.lazy`

By default, `v-model` syncs the input with the data after each `input` event. You can add the `lazy` modifier to instead sync after `change` events:

```
<!-- synced after "change" instead of "input" -->
<input v-model.lazy="msg" >
```

`.number`

If you want user input to be automatically typecast as a number, you can add the `number`modifier to your `v-model` managed inputs:

```
<input v-model.number="age" type="number">
```

`.trim`

If you want user input to be trimmed automatically, you can add the `trim` modifier to your `v-model` managed inputs:

```
<input v-model.trim="msg">
```

Components

Basic

Local registration

Using 'is' for component

Data must be a function

Props (static/dynamic)

Kebab-case

Literal vs dynamic

Prop validation

Custom events

Non Parent-Child Communication

Content Distribution with Slots(single slot/named slots)

Dynamic Components

Keep alive