

Rewriting for that 1000th Use Case

A Vue Component Story



Damian Dulisz

GitHub: @shentao

Twitter: @damiandulisz

Vue.js Core Team

news.vuejs.org

Official Vue.js Newsletter and News Portal

with Gregg Pollack and Adam Jahr

Vuelidate

Simple, lightweight model-based validation for Vue.js

with Paweł Grabarz @frizi

Vue-Global-Events

Register global events as a component

with Eduardo San Martin Morote @posva

Vue-Multiselect

Universal select/multiselect/tagging component for Vue.js

*„Maintaining a popular OSS component library is like maintaining a component for the biggest app in the world, because **every single use case possible** eventually comes up.“*

– Chris Fritz



Vue-multiselect

(2.1.3)

The most complete selecting solution for **Vue.js**

Stars 4k license MIT downloads 136k/m dependencies none

Pick badges

- Single / multiple select
- Dropdowns
- **Searchable**
- **Tagging**
- Server-side Rendering support
- **Vuex support by default**
- **Ajax support**
- **Fully configurable**
- +99% test coverage
- No dependencies

 View on GitHub

Getting started & examples

★ 3,694

GitHub Stars

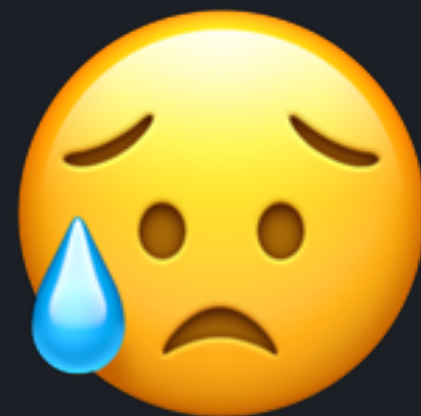
💾 1,250,460

NPM downloads in the last 12 months




! 726

Issues (65 Open)



Not all are bugs!

 726
Issues (65 Open)

{	80	BUGS
	178	QUESTION
	135	ENHANCEMENT
	191	UNLABELED
	136	OTHER LABELS

What I've learned

Sources of bugs

Wrong UX assumptions

Not enough tests

Bad design decisions

Feature creep

Wrong UX assumptions

Not enough te

Wrong UX assumptions

Not enough to

Instead of inventing new interaction patterns,
try to mimic builtin HTML element behaviour.
(or look for popular patterns)

Especially important for keyboard control!

g UX assumptions

Not enough tests

Bad design decisions

~1700 lines of tests

~100 test cases

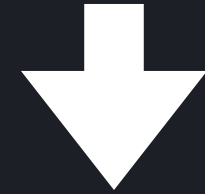
 **Still not enough**

g UX assumptions

Not enough tests

Bad design decisions

More features



More configuration options



More combinations to test

Make sure to test every possible combination of features!

Not enough tests

Bad design decisions

Feature creep

Made back in 2016

Not enough tests

Bad design decisions

Feature creep

Creating local, synchronised copies of props.

```
{
  data () {
    return {
      internalValue: deepClone(this.value)
    }
  },
  watch: {
    value () {
      this.internalValue = deepClone(this.value)
    }
  }
}
```

👉 **Smell**

lot enough tests

Bad design decisions

Feature creep

Creating local, synchronised copies of props.

Avoid local state if possible.

Always derive from props.

```
const x = {  
  computed: {  
    internalValue () {  
      return this.value || this.value === 0  
        ? [].concat(this.value)  
        : []  
    }  
  }  
}
```

✓ **Recommended**

lot enough tests

Bad design decisions

Feature creep

Passing event handlers as props.

```
<VueMultiselect :onUpdate="updateValue"/>
```

```
update (newValue) {  
  if (this.onUpdate) this.onUpdate(newValue)  
}
```

Requires checks if prop exists or a no-op default function.

👉 Smell

Not enough tests

Bad design decisions

Feature creep

Passing event handlers as props.

Use Vue's event system.

```
<VueMultiselect @input="updateValue"/>  
  
update (newValue) {  
  this.$emit('input', newValue)  
}
```

✓ **Recommended**

Not enough tests

Bad design decisions

Feature creep

Using scoped styles.

Having styles as part of the JS bundle.

 **Avoid in OSS**

Not enough tests

Bad design decisions

Feature creep

Using scoped styles.

Having styles as part of the JS bundle.

 **Avoid in OSS**

Do not use scoping.

Use BEM naming convention or similar.

Can use CSS Modules if configured without hash.

Flat class structure.

Extract styles to separate CSS file.

 **Recommended**

Not enough tests

Bad design decisions

Feature creep

Extract styles to separate CSS file.

In webpack based builds use:

[mini-css-extract-plugin](#)

In Rollup based builds use:

[rollup-plugin-postcss](#)

Not enough tests

Bad design decisions

Feature creep

Using mixins to separate concerns.

- Multiselect.vue
- multiselectMixin.js
- pointerMixin.js

 **Complete failure**

Not enough tests

Bad design decisions

Feature creep

Using mixins to separate concerns.

Use components instead.

- MultiselectWrapper.vue
- MultiselectOptions.vue
- MultiselectValue.vue
- MultiselectInput.vue

✓ **Recommended**

Not enough tests

Bad design decisions

Feature creep

Relying on focus and blur events to manage active state

😓 **Still hurts**

Not enough tests

Bad design decisions

Feature creep

Relying on focus and blur events to manage active state

😓 **Still hurts**

Use click-outside and
other explicit user actions like pressing TAB/Shift+TAB

✓ **Recommended**

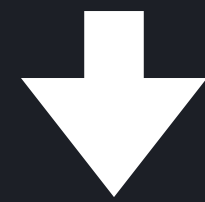
Bad design decisions

Feature creep

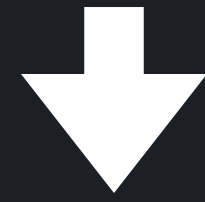
Bad design decisions

Feature creep

More features



More complexity



Harder to maintain and contribute

Bad design decisions *Feature creep*

Instead of adding all the requested features,
try to provide an API that makes it easy to implement
custom functionalities on top of your library



Probably the most **important** lesson here

135

ENHANCEMENT

User needs

85%

Existing features are
enough for their needs

15%

Need more features and
customisation options

Actually just a guess

How to satisfy both groups?

- Without sacrificing the **ease of use**.
- While avoiding **unnecessary complexity**.
- Making **contributions easy**.

How others do it?

React-Select

<https://github.com/JedWatson/react-select>

React-Select

It *just works* out of the box

Extremely customisable

Lots of features

Somewhat opinionated

Large API surface

Downshift



<https://github.com/paypal/downshift>

Downshift

Build your own UI

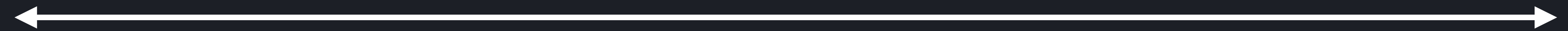
Minimal API

Incredibly flexible

Not ready to use out-of-the-box

React-Select

Downshift



features
complex
large API surface
just works

flexible
simple
minimal API surface
makes you work

How do we handle edge cases?

React-Select

Downshift



in the library

in user code

React-Select

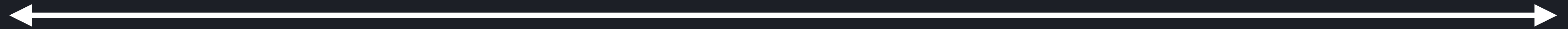
Downshift

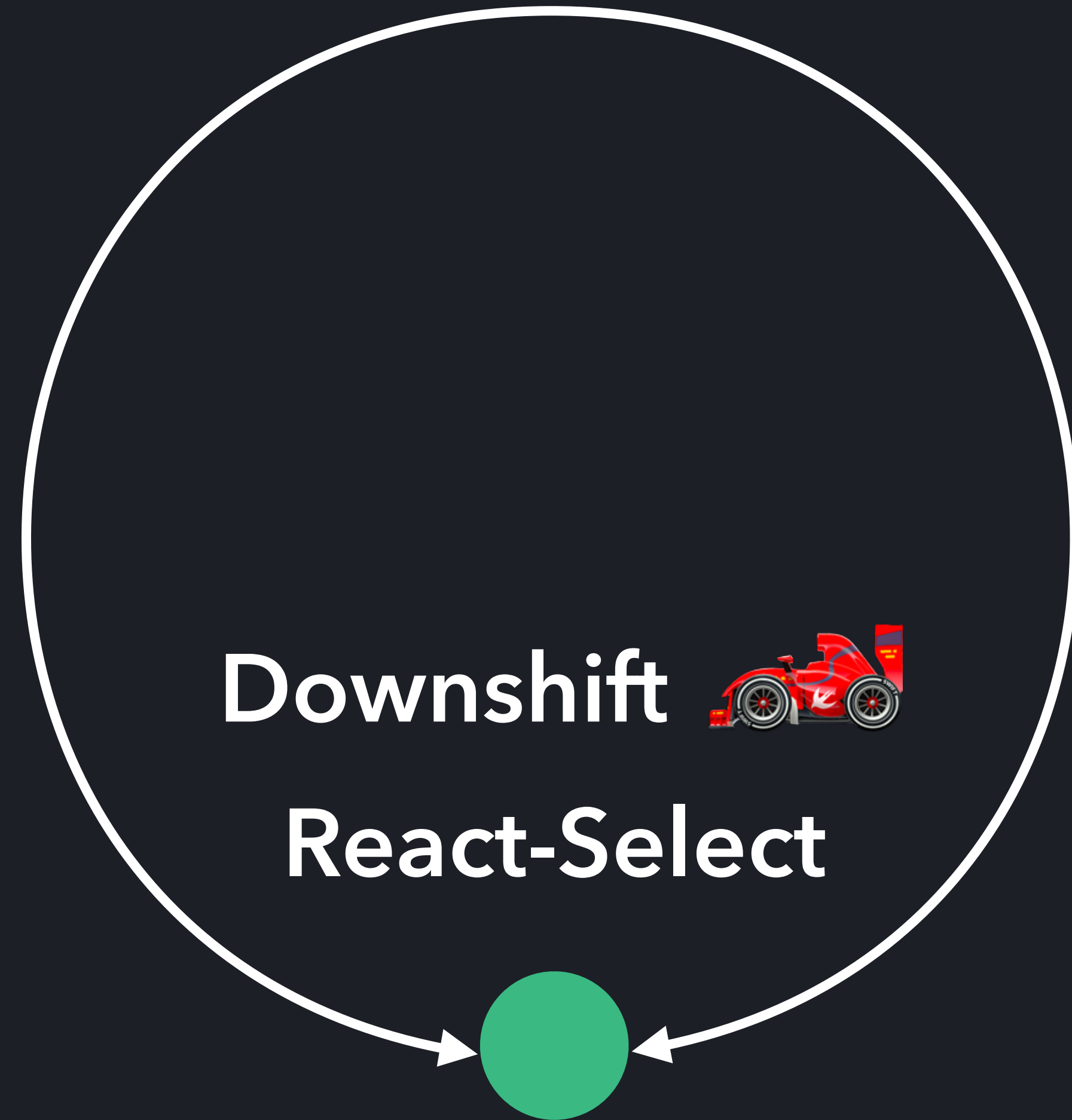


Vue-Multiselect v2.x

React-Select

Downshift







That doesn't make sense

MOVIECLIPS.COM

Except it does

Introducing

Vue-Multiselect v3.0



Vue-Multiselect v3.0

MultiselectCore.js

Renderless component managing the core functionality. Exposes state, methods and computed properties through the default scoped slot.

No UI.

Similar to Downshift 

Using MultiselectCore.js

```
<template>
  <MultiselectCore
    v-bind="$props"
    v-on="$listeners"
  >
    <div slot-scope="{ select, options, value }">
      <!-- Actual component UI -->
    </div>
  </MultiselectCore>
</template>
```

Renderless Component

Forwarding props and listeners

Receiving data and methods in the default slot

Vue-Multiselect v3.0

MultiselectCore.js

Renderless component managing the core functionality. Exposes state, methods and computed properties through the default scoped slot.

No UI.

Vue-Multiselect v3.0

Multiselect.vue

The default composition that implements all features from Vue-Multiselect v2.x.

Proxies all props and event listeners to MultiselectCore.

MultiselectCore.js

Renderless component managing the core functionality. Exposes state, methods and computed properties through the default scoped slot.

No UI.

Works exactly like Vue-Multiselect v2.x

Multiselect.vue

The default composition that implements the features known from Vue-Multiselect v2.x. It proxies all of it's props and listeners to the MultiselectCore component.

MultiselectCore.js

The renderless core component containing all VueMultiselect logic. It exposes all the data, computed properties and methods to all of its children through a slot scope.

— DEFAULT SLOT —

— DEFAULT SLOT —

MultiselectWrapper.vue

The root component responsible for positioning the component as well as handling focus and keyboard events.

— CONTROL —

MultiselectValue.vue

The component responsible for displaying the selected results. It contains multiple slots like: **tag**, **loading**, **caret**, **singleLabel**, **placeholder** and **control**.

MultiselectInput.vue

Text input component used for changing the search value. Present only when the dropdown is open. When present, it controls the keyboard events.

— OPTIONS —

MultiselectOptions.vue

Component responsible for the list of options and the way they are displayed and navigated. Handles mouse clicks and highlights elements based on the pointer value and selected results.

```
// Multiselect.vue
<template>
  <MultiselectCore
    v-bind="$props"
    v-on="$listeners"
  >
    <div
      slot-scope="scope"
      style="position: relative;"
    >
      <MultiselectWrapper v-bind="scope">
        <MultiselectValue
          v-bind="scope"
          :class="{
            'multiselect--disabled': scope.disabled
          }"
        >
          <MultiselectInput
            v-if="scope.searchable"
          >
```

What's the gain?

What's the gain?

- Creating new compositions based on the Core with completely new features and UI
- Sharing new partial components that can be used as plug-in replacements for existing elements
- Easier contribution, more maintainable code

Current status: pre-alpha

Works, but more changes are coming

Try it out anyway

<https://github.com/shentao/vue-multiselect/tree/v3>

Feedback appreciated!

Bonus

<https://github.com/chrisvfritz/hello-vue-components>

Thank you!

Damian Dulisz

GitHub: @shentao

Twitter: @damiandulisz

Questions?

Renderless component

- Looks almost like a regular component
- Does not have a template or styles, just JavaScript
- Exposes all of its state and methods to the default slot

```
// MultiselectCore.js
render () {
  return this.$scopedSlots.default(this)
},
```

Slots Tunneling

Slots inside other components slots, put in a slot

Slots Tunneling

```
<template slot="_option" slot-scope="props">  
  <slot name="option" v-bind="props">  
    <span>{{ getOptionLabel(props.option) }}</span>  
  </slot>  
</template>
```