Vue 3.0 for Library Authors

Damian Dulisz

GitHub: @shentao

Twitter: @damiandulisz

Vue.js Core Team

news.vuejs.org

#140 ## Tuesday, May 7, 2019

Vuex v3.1.1 released. Vue-CLI v4.0-alpha is here; Managing state with Apollo and more!

Hello!

Whoa, there is so much content this week we had a really hard time picking up the articles (some of them might show up next week).

First of all, Vuex got a new release which will definitely make the NativeScript devs happy, since now it supports debugging with the remote Vue Devtools. 🎉

The Vue-CLI v4.0.0-alpha has also been released with several meaningful changes like bumping the versions on webpack-chain, core-js, ESLint, and workbox in the PWA plugin. Take it for a spin and let us know of any issues.

Also, Natalia has written an amazing article on managing state when using Apollo and is probably a must read those of you considering using Apollo in your project.

Cheers,

— Damian Dulisz

PLAY THE PODCAST

Open Source Work

Vue-Multiselect

Universal select/multiselect/tagging component for Vue.js

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

Lead Frontend Engineer

@Coursedoginc

Thanks to all my supporters!

https://github.com/users/shentao/sponsorship

Vueconf.US

NativeScript

Vue Mastery

VueJobs

Vue.js 3.0

for library authors

What is changing?

Vue.js 3.0
for library authors

- Render function changes
  - `h` becomes a globally imported function
  - Unification of render function arguments
- Custom directives API change
  - Aligning it with the new Component lifecycle
- Attribute fall-through changes
  - `class`, `style`, `v-on` and custom directives are all included inside `$attrs`
  - `$listeners` will be removed

Kind of booooooring...

Kind of booooooring...

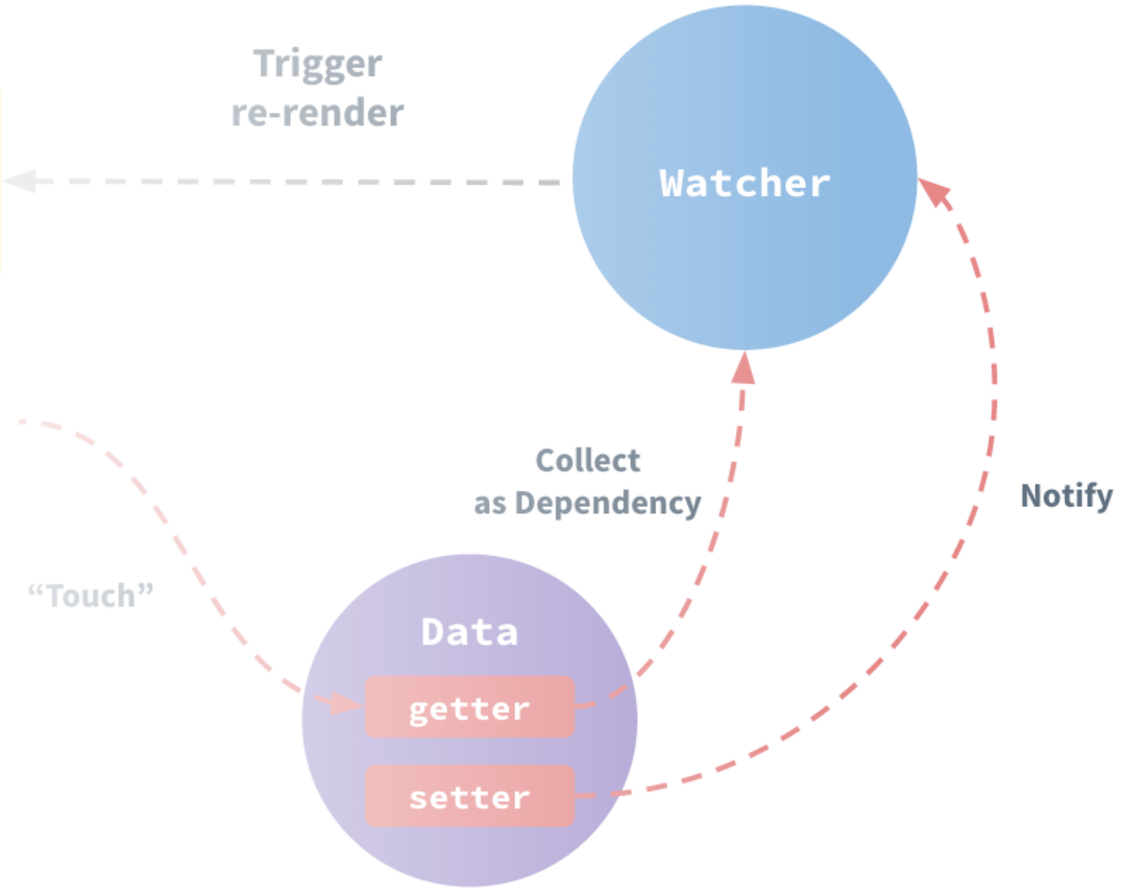
Those are just breaking changes...

Composition API

Composition API = **Advanced Reactivity API** + **Dynamic Lifecycle Injection**

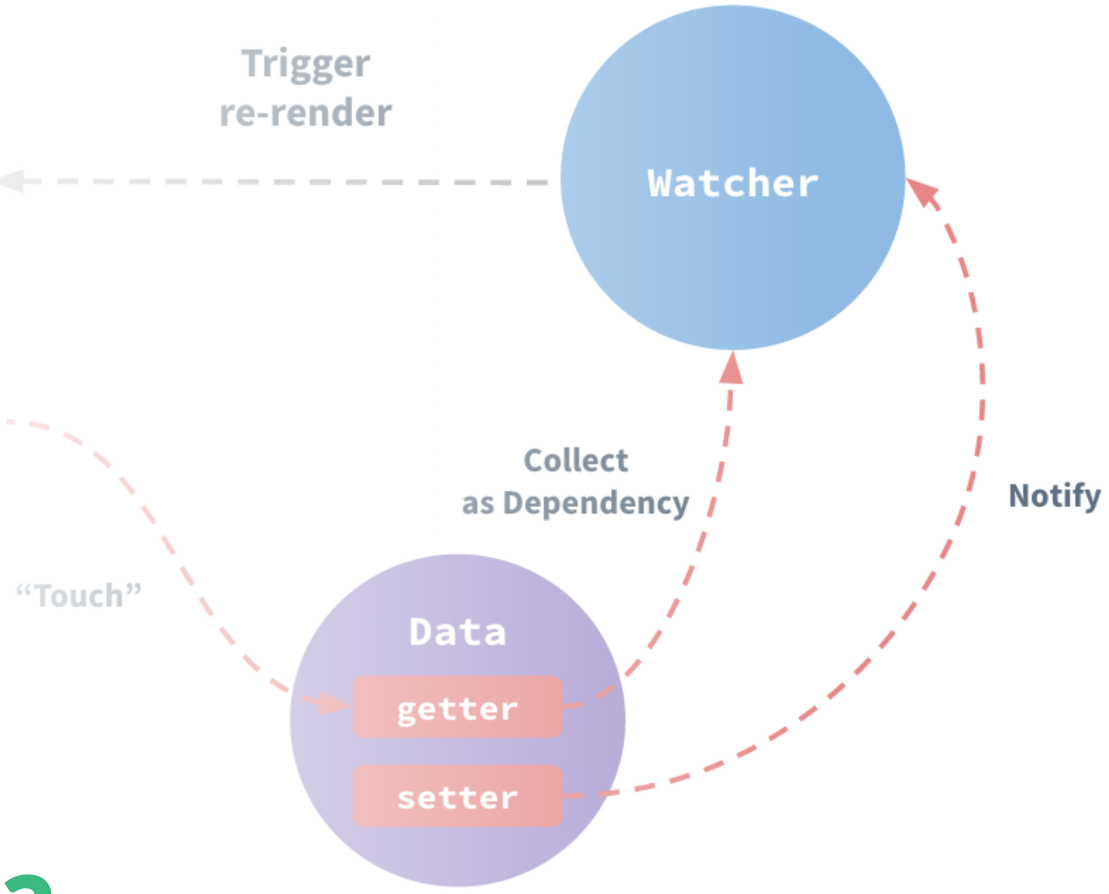
Advanced Reactivity API

- Provides APIs for creating, observing and reacting to reactive state



Advanced Reactivity API

- Provides APIs for creating, observing and reacting to reactive state



Is this truly new though?

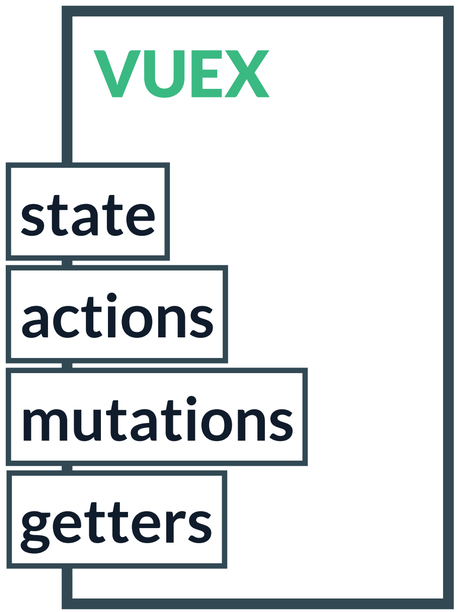
Using the reactivity in Vue v2.0 plugins

Using the reactivity in Vue v2.0 plugins

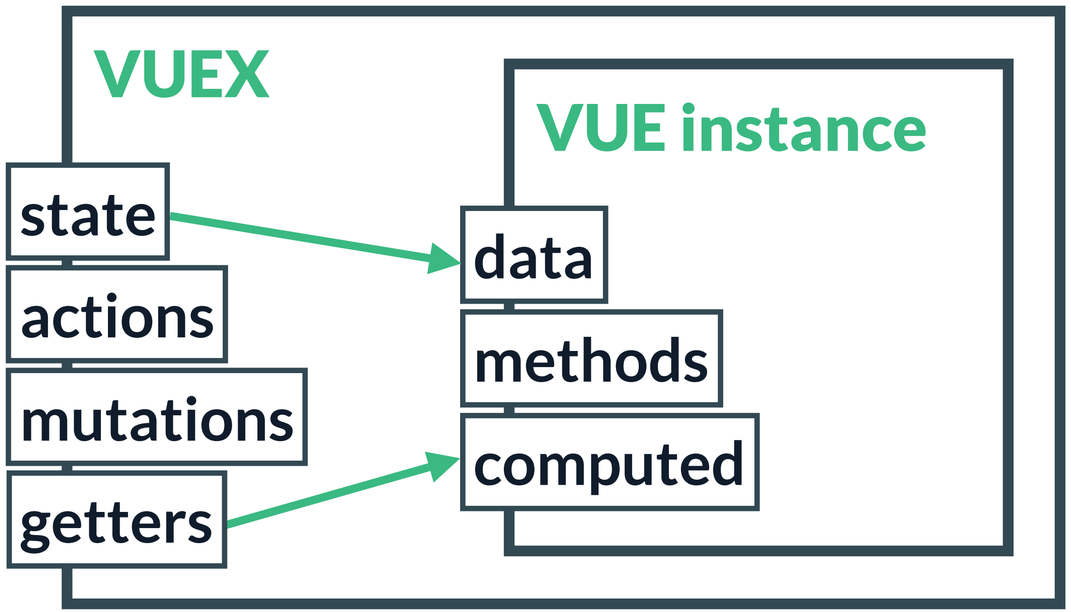
- Through a mixin
  - Extends the current component instance with reactive data and computed properties
- Through creating a new Vue instance
  - Allows for dynamically creating reactive data and computed properties

Ever heard of Vuex?

Ever heard of Vuex?

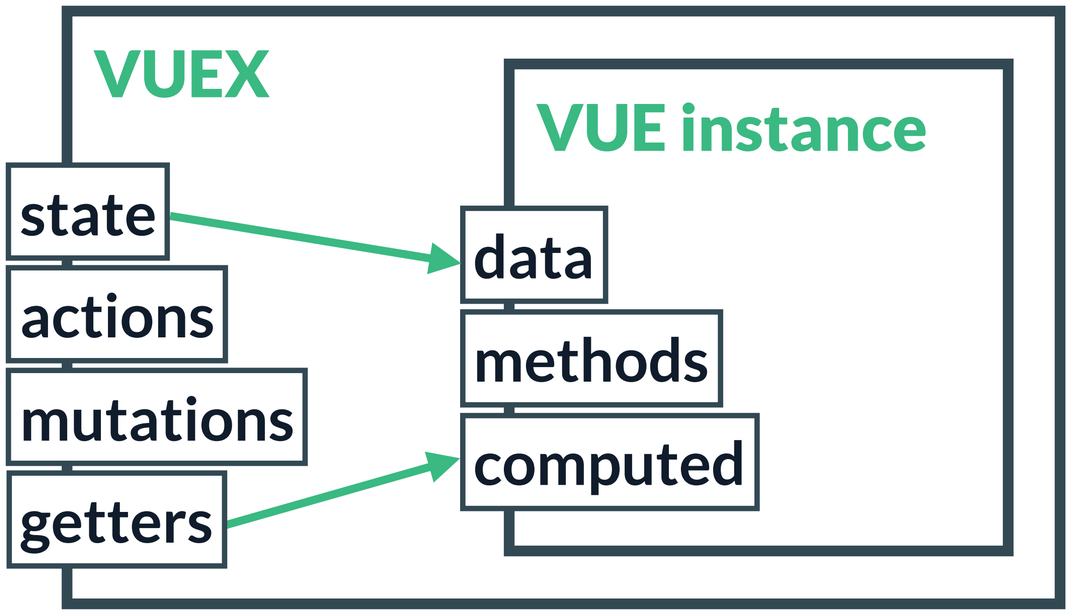


Ever heard of Vuex?



*simplification

Ever heard of Vuex?



*simplification

```
// https://github.com/vuejs/vuex/blob/dev/src/store.js#L273

store._vm = new Vue({
  data: {
    $$state: state
  },
  computed // getters
})
```

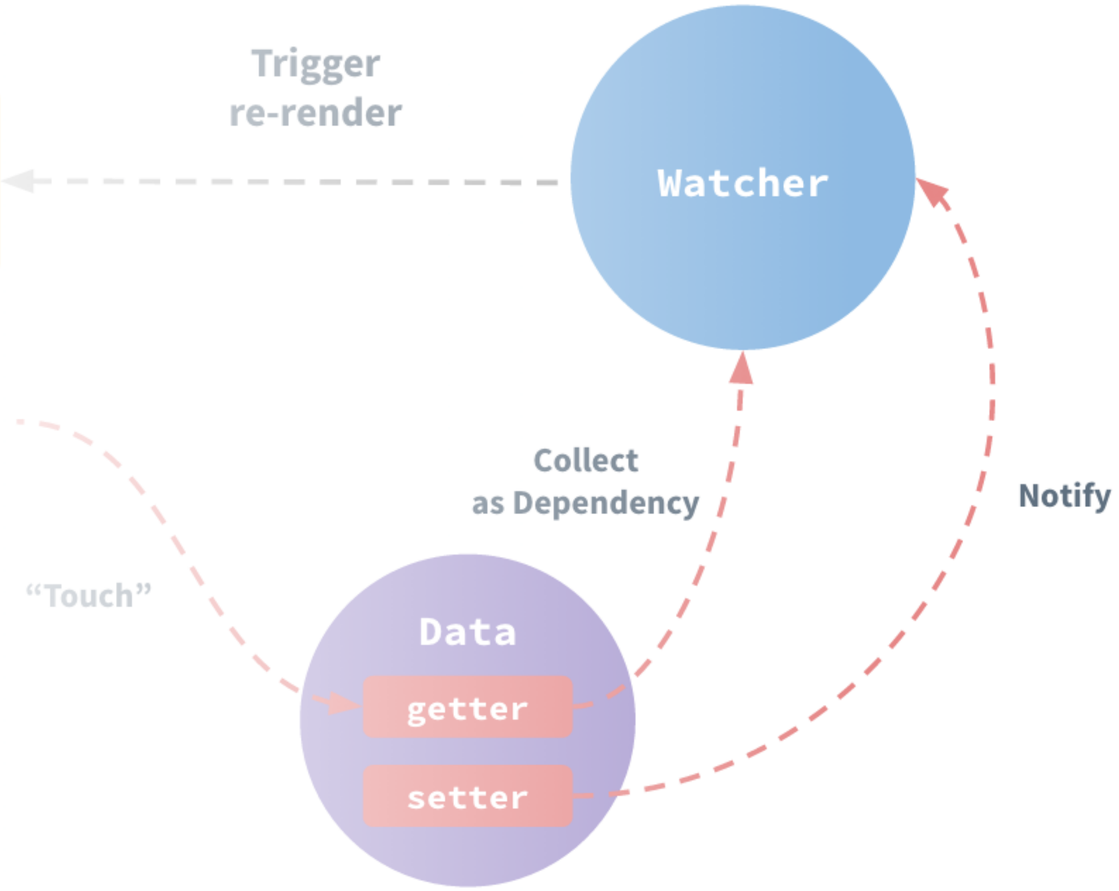
But this isn’t very developer friendly

But this isn’t very developer friendly

- Possible namespace collisions when creating properties using a mixin
- Added overhead due to spawning new Vue instances
- No easy way to create computed properties on the fly
- Not very intuitive

Advanced Reactivity API

- Provides APIs for creating **standalone** reactive state, computed and watchers



Example

Example

[logaretm/vue-use-web](https://github.com/logaretm/vue-use-web)

[shuidi-fed/vue-composition-toolkit](https://github.com/shuidi-fed/vue-composition-toolkit)

Wrapper for WebSocket API

```
// Adapted from
// https://github.com/logaretm/vue-use-web
// /blob/master/src/WebSocket.ts
import {
  ref, onMounted, onUnmounted
} from '@vue/composition-api';

export function useWebSocket (url) {
  const data = ref(null)
  const state = ref('CONNECTING')
  let ws // our websocket

  const close = (code, reason) => {
    if (!ws) return
    ws.close(code, reason)
  }

  const send = (data) => {
    if (!ws) return
    ws.send(data)
  }

  onMounted(() => {
    ws = new WebSocket(url)

    ws.onopen = () => {
      state.value = 'OPEN'
```

Example

[logaretm/vue-use-web]

[shuidi-fed/vue-composition-toolkit]

Wrapper for WebSocket API

```
    ws.send(data)
  }

  onMounted(() => {
    ws = new WebSocket(url)

    ws.onopen = () => {
      state.value = 'OPEN'
    }

    ws.onclose = ws.onerror = () => {
      state.value = 'CLOSED'
    }

    ws.onmessage = (e) => {
      data.value = e.data
    }
  })

  onUnmounted(() => { ws.close() })

  return {
    data,
    state,
    close,
    send
  }
}
```

Applies to building wrappers around other libraries

Think RxJS, real-time messaging, analytic tools and other browser APIs

What existing library could make use of all of this?

Vuelidate!

Vuelidate

November 2016

Started as a single computed property.

```
Vue.mixin({
  beforeCreate() {
    if (!this.$options.validations) return
    const validations = this.$options.validations

    if (typeof this.$options.computed === 'undefined') {
      this.$options.computed = {}
    }

    this.$options.computed.$v = () => {
      return Object.keys(validations)
        .reduce(($validations, key) => {
          const rules = validations[key].call(this)
          $validations[key] = Object.keys(rules)
            .reduce((results, rule) => {
              results[rule] = rules[rule](this[key])
              return results
            }, {})
          return $validations
        }, {})
    }
  }
})
```

Vuelidate

November 2016
Started as a single computed property.

December 2016
Async Validators

```
function makePendingAsyncVm(Vue, promise) {
  const asyncVm = new Vue({
    data: {
      pending: true,
      value: false
    }
  })

  promise
    .then(value => {
      asyncVm.pending = false
      asyncVm.value = value
    }, error => {
      asyncVm.pending = false
      asyncVm.value = false
      throw error
    })

  asyncVm.__isVuelidateAsyncVm = true
  return asyncVm
}
```

Vuelidate

November 2016
Started as a single computed property.

December 2016
Async Validators

2017-2018
Countless new features later

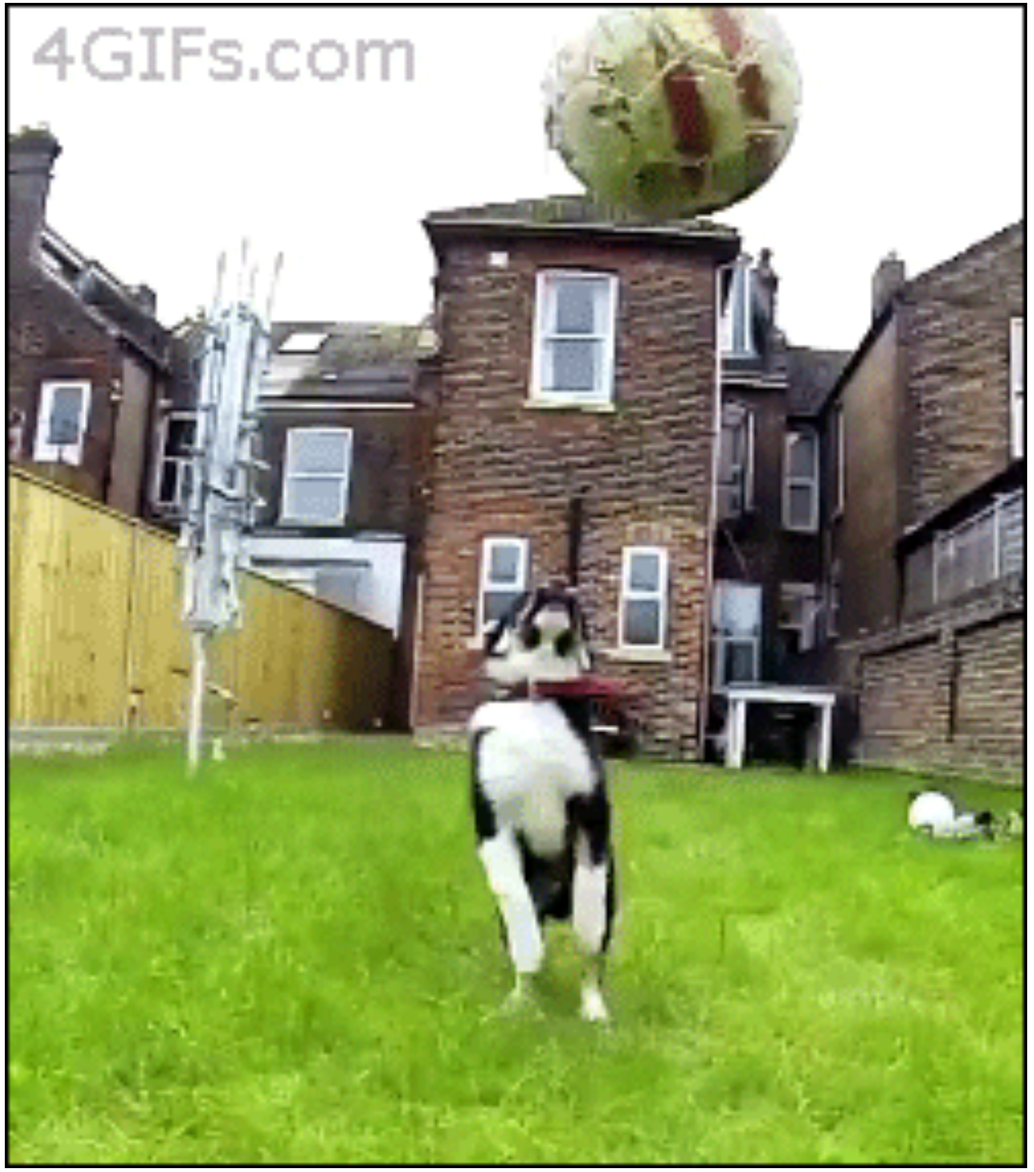
```
const validateModel = (model, validations) => {
  const Vue = getVue(model)
  const { Validation, VBase } = getComponent(Vue)
  const root = new VBase({
    computed: {
      children() {
        const vals =
          typeof validations === 'function'
            ? validations.call(model)
            : validations

        return [
          h(Validation, '$v', {
            validations: vals,
            lazyParentModel: NIL,
            prop: '$v',
            model,
            rootModel: model
          })
        ]
      }
    }
  })
  return root
}
```

The code got very complicated and hard to maintain

Let’s see how we can change that.

```
yarn add @vue/composition-api
```


4GIFs.com

The only utils function you will ever need

The only utils function you will ever need

```
import { isRef } from ‘@vue/composition-api'

export const unwrap = v => isRef(v) ? v.value : v
```

Vuelidate

Creating a synchronous validation result

```
function createComputedResult (rule, model) {
  return computed(() => {
    const result = callRule(rule, model)
    return result.$invalid !== undefined
      ? result.$invalid
      : !result
  })
}
```

Vuelidate

Creating an async validation result

```
function createAsyncResult (
  rule, model, $pending
) {
  const $invalid = ref(true)
  $pending.value = true

  watch(
    model,
    modelValue => {
      const ruleResult = callRule(rule, modelValue)
      $pending.value = true
      $invalid.value = true

      ruleResult
        .then(data => {
          $pending.value = false
          $invalid.value = !data
        })
        .catch(() => {
          $pending.value = false
          $invalid.value = true
        })
    },
    { lazy: true }
  )
  return $invalid
}
```

Pretty cool, right?

Vue.js 3.0

Allows you to focus on **what**
you want to create,
rather than on **how** to do it.

That’s it…?

Vuelidate v2.0 alpha

vuelidate/vuelidate/tree/next

```
yarn add @vuelidate/core @vuelidate/validators
```

Vuelidate v2.0

- Written from scratch
- Built for Vue 3.0
- Test-drive with Vue 2.x (with `@vue/composition-api`)
- Half the code, quarter the complexity
- Mostly backwards compatible
- Some breaking changes though…

Vuelidate v2.0

Basic component with `password` and `repeatPassword` in the state.

```
export default {
  setup () {
    const password = ref('')
    const repeatPassword = ref('')

    return {
      password,
      repeatPassword,
    }
  }
}
```

Vuelidate v2.0

Let’s import our new Vuelidate composition function `useVuelidate`.

It accepts and object containing the rules and an object containing the state that the rules will validate.

```
import useVuelidate from ‘@vuelidate/core'

export default {
  setup () {
    const password = ref('')
    const repeatPassword = ref('')
    const $v = useVuelidate(
      rules,
      state
    )
    return {
      password,
      repeatPassword,
      $v
    }
  }
}
```

Vuelidate v2.0

The rules, similar to Vuelidate 0.x should map 1:1 to the state that will be passed.

```
import useVuelidate from '@vuelidate/core'

export default {
  setup () {
    const password = ref('')
    const repeatPassword = ref('')
    const $v = useVuelidate(
      {
        password: {
          required,
          minLength: minLength(7),
        },
        repeatPassword: {
          sameAs: sameAs(password),
        }
      },
      {
        password,
        repeatPassword
      }
    )

    return {
      password,
      repeatPassword,
      $v
    }
  }
```

Vuelidate v2.0

The rules, similar to Vuelidate 0.x should map 1:1 to the state that will be passed.

What about backwards compatibility?

Vuelidate v2.0

What about backwards compatibility?

Just a few lines of code!

```
// Vuelidate 2.0 source
export const VuelidateMixin = {
  beforeCreate () {
    const options = this.$options
    if (!options.validations) return

    const validations = isFunction(
      options.validations
    )
      ? options.validations.call(this)
      : options.validations

    if (!options.computed) options.computed = {}
    if (options.computed.$v) return

    // this is where the magic happens
    options.computed.$v = () => setValidations({
      validations,
      state: this
    })
  }
}
```

Vuelidate v2.0

What about backwards compatibility?

Just a few lines of code!

```
export default {
  mixins: [VuelidateMixin],
  data () {
    return {
      password: '',
      repeatPassword: ''
    }
  },
  validations () {
    return {
      password: {
        required,
        minLength: withMessage(
          minLength(7),
          ({ $params }) =>
            `Has to be at least ${$params.length}
             characters long`
        ),
      },
      repeatPassword: {
        sameAs: withMessage(
          sameAs(this.password),
          'Doesn’t match the password'
        ),
      }
    }
  }
}
```

Vuelidate v2.0

`withMessage`

A function that adds a message (string or function) to a validator.

```
minLength: withMessage(
  minLength(7),
  ({ $params }) =>
    `Has to be at least ${$params.length}
     characters long`
),
```

Vuelidate v2.0

Looks like a lot of work?

Vuelidate v2.0

Looks like a lot of work?

Not necessarily!

Vuelidate v2.0

Looks like a lot of work?

Not necessarily!

All 21 built-in validators come with messages and params already set up.

Vuelidate v2.0

All 21 built-in validators come with messages and params already set up.

```
alpha
alphaNum
and
between
decimal
email
integer
ipAddress
macAddress
maxLength
maxValue
minLength
minValue
not
numeric
or
required
requiredIf
requiredUnless
sameAs
url
```

Vuelidate v2.0

```
alpha
alphaNum
and
between
decimal
email
integer
ipAddress
macAddress
maxLength
maxValue
minLength
minValue
not
numeric
or
required
requiredIf
requiredUnless
sameAs
url
```

You choose which one to import from `@vuelidate/validators`.
Fully tree-shakeable.

Vuelidate v2.0

Build your own validators!

```
import minLength from '../raw/minLength'

export default length => ({
  $validator: minLength(length),
  $message: ({ $params }) => `This field should be at
    least ${$params.length} long.`,
  $params: { length }
})
```

Submit a **Pull Request** to share it with others!

Vuelidate v2.0

Demo Time!

Vuelidate v2.0

Goodbye

`vuelidate-error-extractor` 👋

It might become a UI helper for errors though

Sorry Dobromir 😅

Vuelidate v2.0

Breaking changes

Vuelidate v2.0

Breaking changes

Removing $each schema field

Vuelidate v2.0

Breaking changes

Removing `$each` schema field

Instead use `every` or `some` helpers that accept a validator.

Vuelidate v2.0

Instead use `every` or `some` helpers that accept a validator.

```
const $v = useVuelidate({
  emailList: {
    everyEmail: every(email),
  }
}, { emailList })
```

Vuelidate v2.0

Or move the validators to the child component

Vuelidate v2.0

Roadmap:

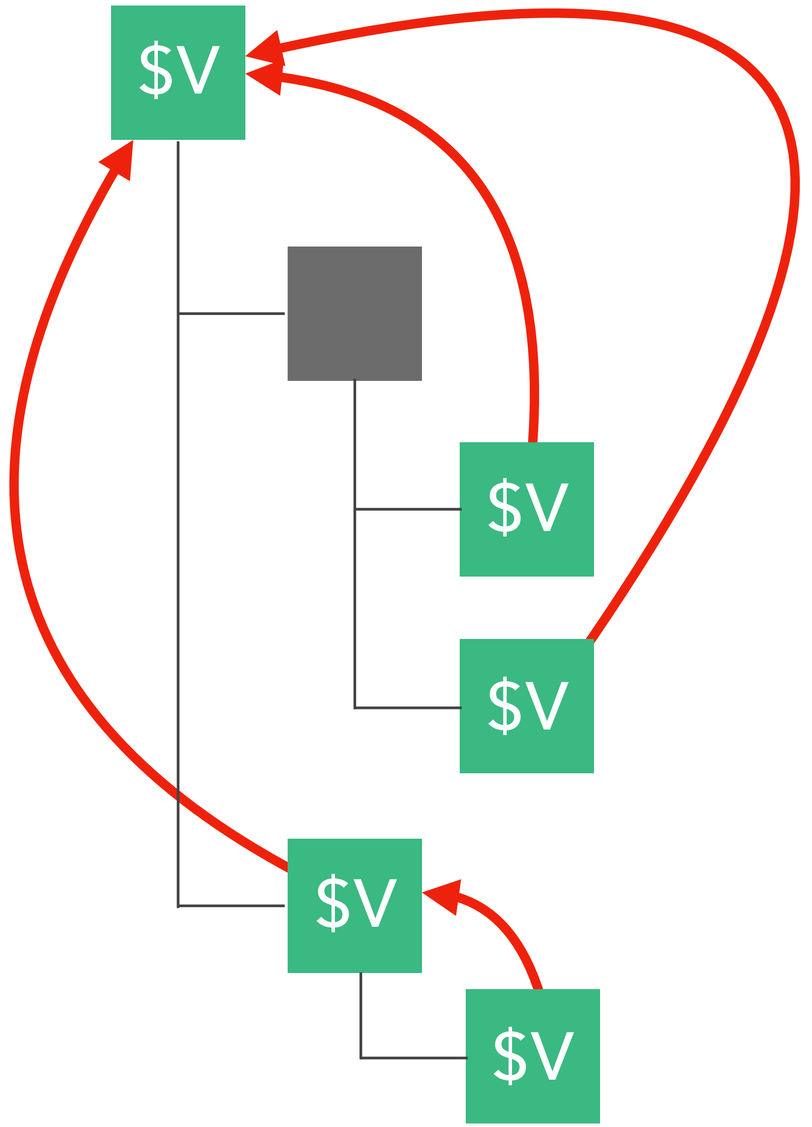
- Context aware validations
- i18n support
- Lazy validation
- Better server validation
- Migration to TypeScript
- *Your idea?*

Vuelidate v2.0

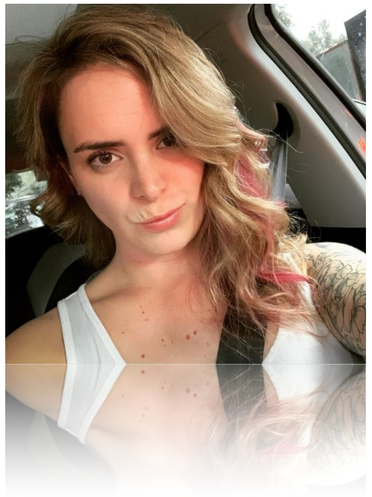
Context aware validations

Child components with validations will report back to the ancestor component.

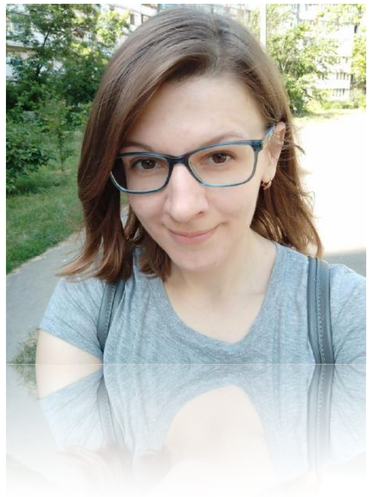
Meaning, the root form component will know the validation results of all child components.



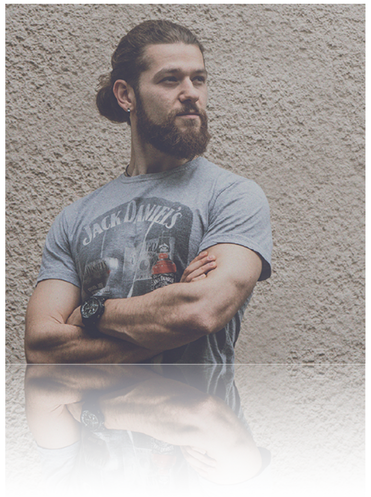
Meet the Vuelidate team!



Marina Mosti
@marinamosti



Natalia Tepluhina
@n_tepluhina



Dobromir Hristov
@d_m_hristov

Ben Hong
@bencodezen

That guy on the stage

One more thing...

FORMVUELATTE

Lightweight Schema Generated Forms

vuelidate/formvuelatte

FORMVUELATTE

Lightweight Schema Generated Forms

- “Bring Your Own Components”
- Less than 100 LoC

Roadmap:

- Opt-in integration with Vuelidate
- Adapters for popular UI libraries like Vuetify

Thank You!

Damian Dulisz

GitHub: @shentao

Twitter: @damiandulisz

Vue.js Core Team

https://github.com/users/shentao/sponsorship

Vuelidate v2.0

All validator-named properties are now objects containing their validation results:
`$invalid, $message, $params, $pending`

`$v.password` →

```
"password": {
  "$dirty": false,
  "$pending": false,
  "required": {
    "$message": "This field is required",
    "$pending": false,
    "$invalid": true
  },
  "minLength": {
    "$message": "Has to be at least 7 characters long",
    "$params": {
      "length": 7
    },
    "$pending": false,
    "$invalid": true
  },
  "$invalid": true,
  "$error": false,
  "$errors": [
    {
      "$property": "password",
      "$validator": "required",
      "$message": "This field is required",
      "$pending": false
    },
    {
      "$property": "password"
```

Vuelidate v2.0

Introducing `$errors`, a list of all failing validators.

`$v.password` →

```
    },
    "$pending": false,
    "$invalid": true
  },
  "$invalid": true,
  "$error": false,
  "$errors": [
    {
      "$property": "password",
      "$validator": "required",
      "$message": "This field is required",
      "$pending": false
    },
    {
      "$property": "password",
      "$validator": "minLength",
      "$message": "Has to be at least 7 characters long",
      "$params": {
        "length": 7
      },
      "$pending": false
    }
  ],
  "$model": "",
  "$anyDirty": false
},
```

Vuelidate v2.0

Just like other meta-properties, `$errors` are being collected and merged from the nested leaf properties back to the root.

Meaning `$v.$errors` will contain the list of all current errors with messages!

`$v` →

```
"$errors": [
  {
    "$property": "password",
    "$validator": "required",
    "$message": "This field is required",
    "$pending": false
  },
  {
    "$property": "password",
    "$validator": "minLength",
    "$message": "Has to be at least 7 characters long",
    "$params": {
      "length": 7
    },
    "$pending": false
  },
  {
    "$property": "repeatPassword",
    "$validator": "required",
    "$pending": false
  }
]
```