



# Vue 3.0 for Library Authors



**Damian Dulisz**

GitHub: [@shentao](#)

Twitter: [@damiandulisz](#)

Vue.js Core Team



## #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

### The Official Vue News

We exist to provide Vue developers with the latest news and tutorials to stay up-to-date with their technology.

#### Get a weekly email

 Email addressSUBSCRIBE

Previously known as Vue.js Newsletter

#### Subscribe to the podcast

[APPLE PODCASTS](#)[ANDROID](#)[STITCHER](#)[RSS](#)

# Open Source Work

## Vue-Multiselect

Universal select/multiselect/tagging component for Vue.js

## Vuelidate

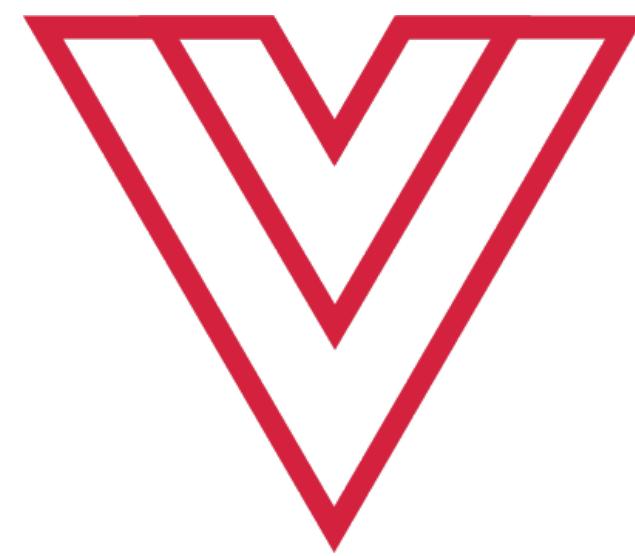
Simple, lightweight model-based validation for Vue.js

with Paweł Grabarz @frizi

## Vue-Global-Events

Register global events as a component

with Eduardo San Martin Morote @posva



# Vue.js Poland

## Facebook Group:

<https://www.facebook.com/groups/vuejspolska/>

- ❖ **4543 members**
- ❖ **~500 monthly posts/comments/reactions**

## Meetups:

<https://www.meetup.com/Vue-js-Poland/events/>

- ❖ **9 meetups in 4 cities to date**
- ❖ **1394 members**
- ❖ **~80 attendees on average**



**Lead Frontend Engineer**  
**@Coursedoginc**

# Thanks to all my supporters!

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

Vueconf.US



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 booooring...**  
**Those are just breaking  
changes...**



# Composition API

**Composition  
API**

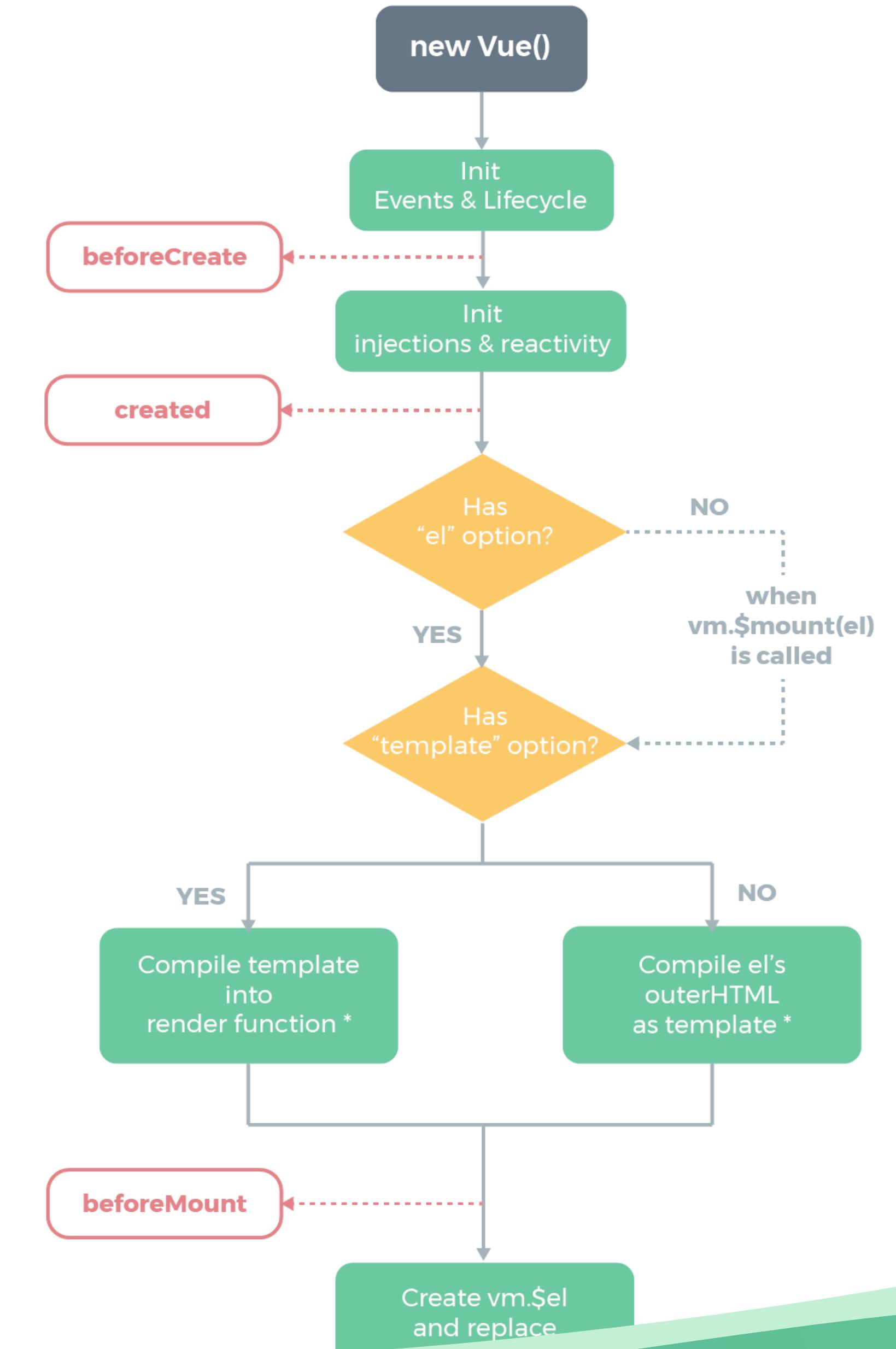


**Advanced  
Reactivity +  
API**

**Dynamic  
Lifecycle  
Injection**

# Dynamic Lifecycle Injection

- ❖ Allows to dynamically add and remove lifecycle hooks

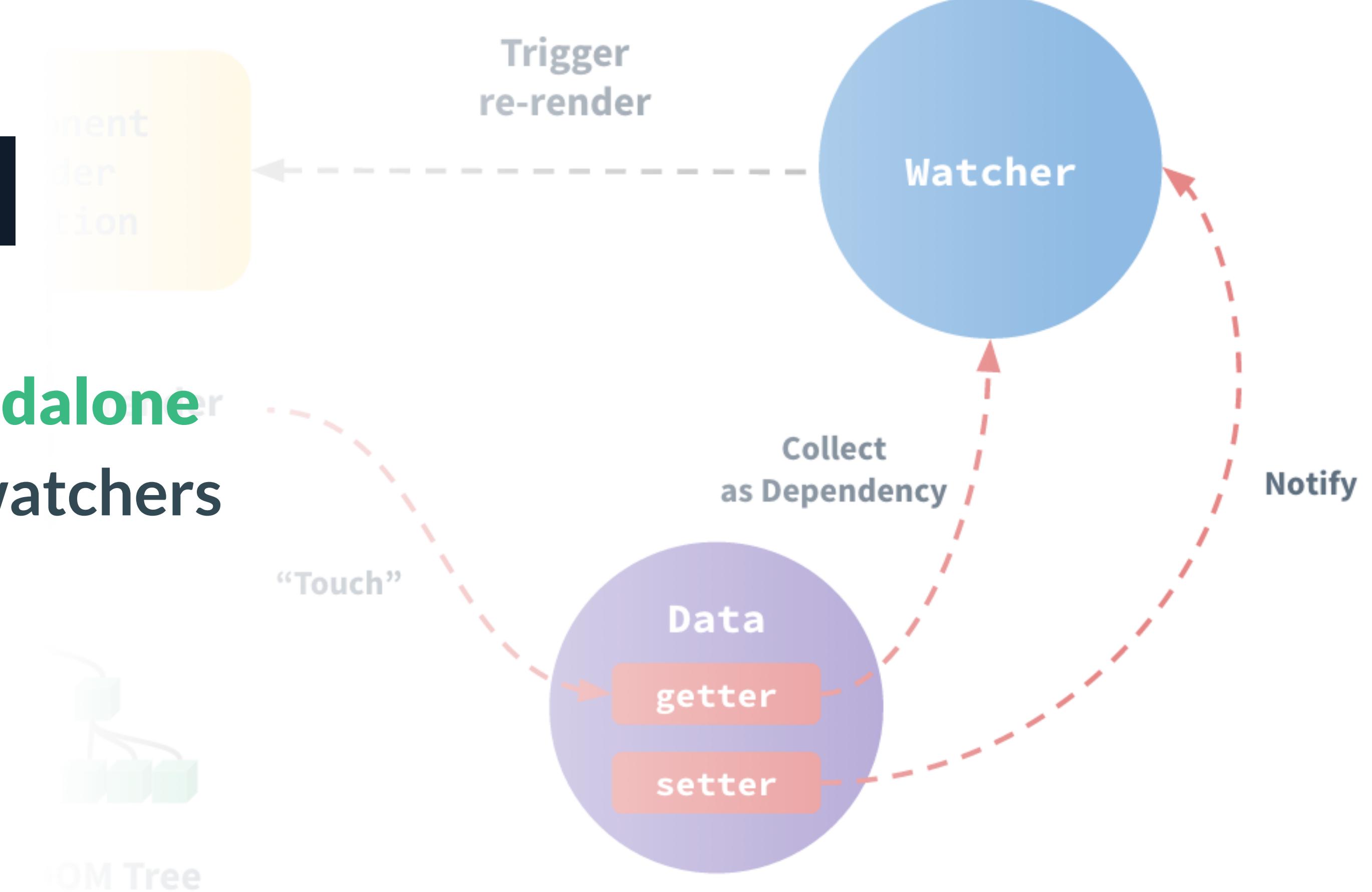


# Dynamic Lifecycle Injection

```
import {  
  onMounted, onUpdated, onUnmounted  
} from '@vue/composition-api'  
  
// inside a Vue component or composition function  
export default {  
  setup () {  
    onMounted(() => {  
      console.log('mounted!')  
    })  
  
    onUpdated(() => {  
      console.log('updated!')  
    })  
  
    onUnmounted(() => {  
      console.log('unmounted!')  
    })  
  }  
}
```

# Advanced Reactivity API

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



# Advanced Reactivity API

```
import {  
    reactive  
} from '@vue/composition-api'  
  
// create a reactive object  
const state = reactive({ counter: 1 })
```

# Advanced Reactivity API

```
import {  
    reactive, ref  
} from '@vue/composition-api'  
  
// create a reactive object  
const state = reactive({ counter: 1 })  
  
// create an object wrapper around a  
// primitive value which acts as a pointer  
// that can be passed by reference  
const counter = ref(0)  
  
// read the pointer's value  
console.log(counter.value) // 0  
  
// mutate the value  
counter.value++
```

# Advanced Reactivity API

```
import {  
    reactive, ref, computed  
} from '@vue/composition-api'  
  
// create an object wrapper around a  
// primitive value which acts as a pointer  
// that can be passed by reference  
const counter = ref(0)  
  
// create a computed pointer  
const counterPlusOne = computed(  
    () => counter.value + 1  
)
```

# Advanced Reactivity API

```
import {
  reactive, ref, computed, watch
} from '@vue/composition-api'

// create an object wrapper around a
// primitive value which acts as a pointer
// that can be passed by reference
const counter = ref(0)

// create a computed pointer
const counterPlusOne = computed(
  () => counter.value + 1
)

// pointers can be watched directly
watch(counter, (newVal) => {
  console.log(`counter is: ${newVal}`)
})

watch(counterPlusOne, (newVal) => {
  console.log(`counter plus one is: ${newVal}`)
})
```

# Advanced Reactivity API

## Why is this exciting?



- ❖ Entirely new ways to build Vue plugins that make heavy use of reactive data
- ❖ Easier integration with 3rd-party SDKs and libraries
- ❖ Opens the way for completely new Vuex alternatives

# 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 = () => {
```

# 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
}
```

# Example

[logaretm/vue-use-web](#)

[shuidi-fed/vue-composition-toolkit](#)

Wrapper for WebSocket API

```
// usage in a component
import useWebSocket from './useWebSocket'

export default {
  setup (props) {
    const {
      data,
      status,
      close,
      send
    } = useWebSocket(props.socketUrl)

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

# Advanced Reactivity API

Is this truly new though?

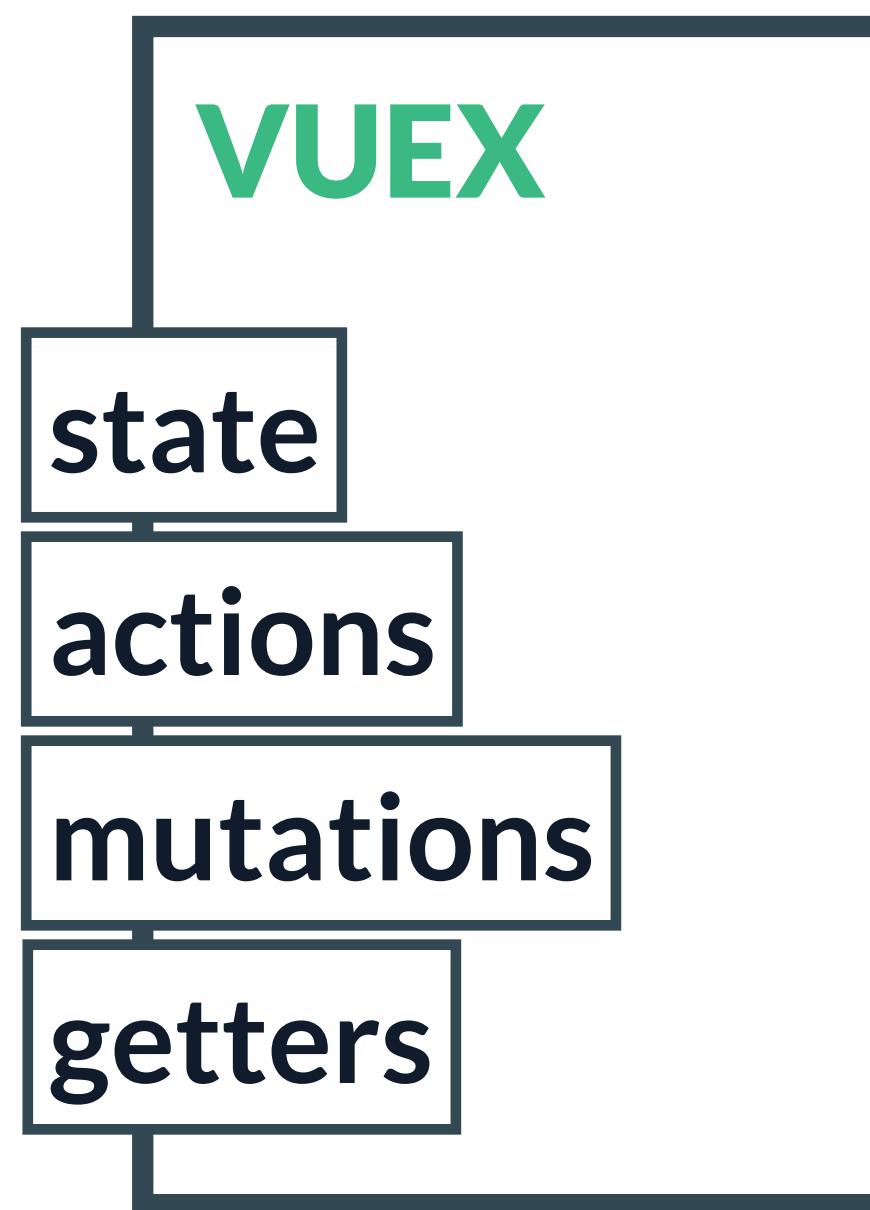
# **Using the reactivity in Vue v2.0 plugins**

# Using the reactivity in Vue v2.0 plugins

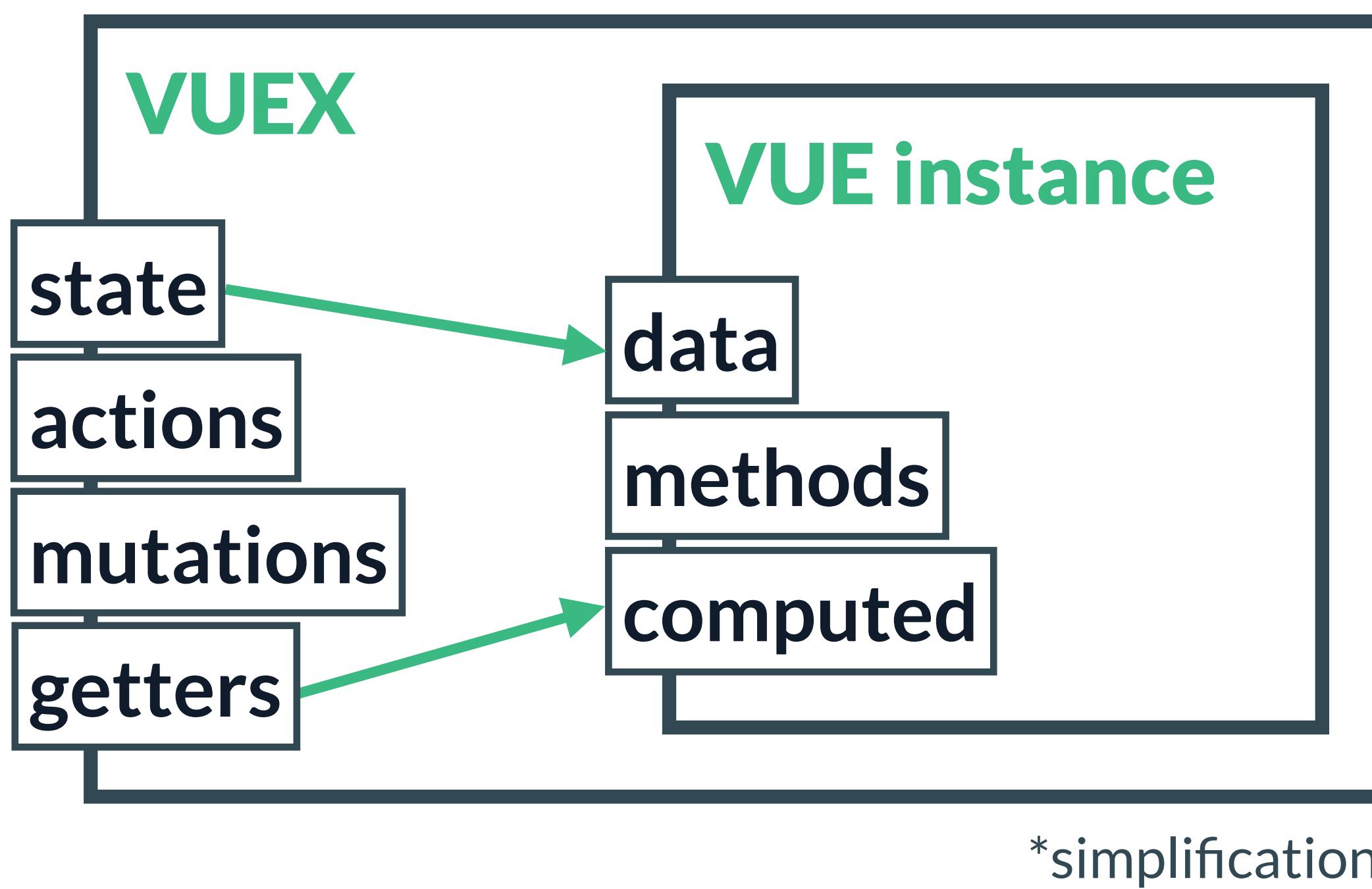
- ❖ Through a mixin
  - ❖ Extends the current component instance with reactive data and computed properties
- ❖ Through creating a new Vue instance
  - ❖ Allows for dynamically creating reactive data and computed properties

# **Ever heard of Vuex?**

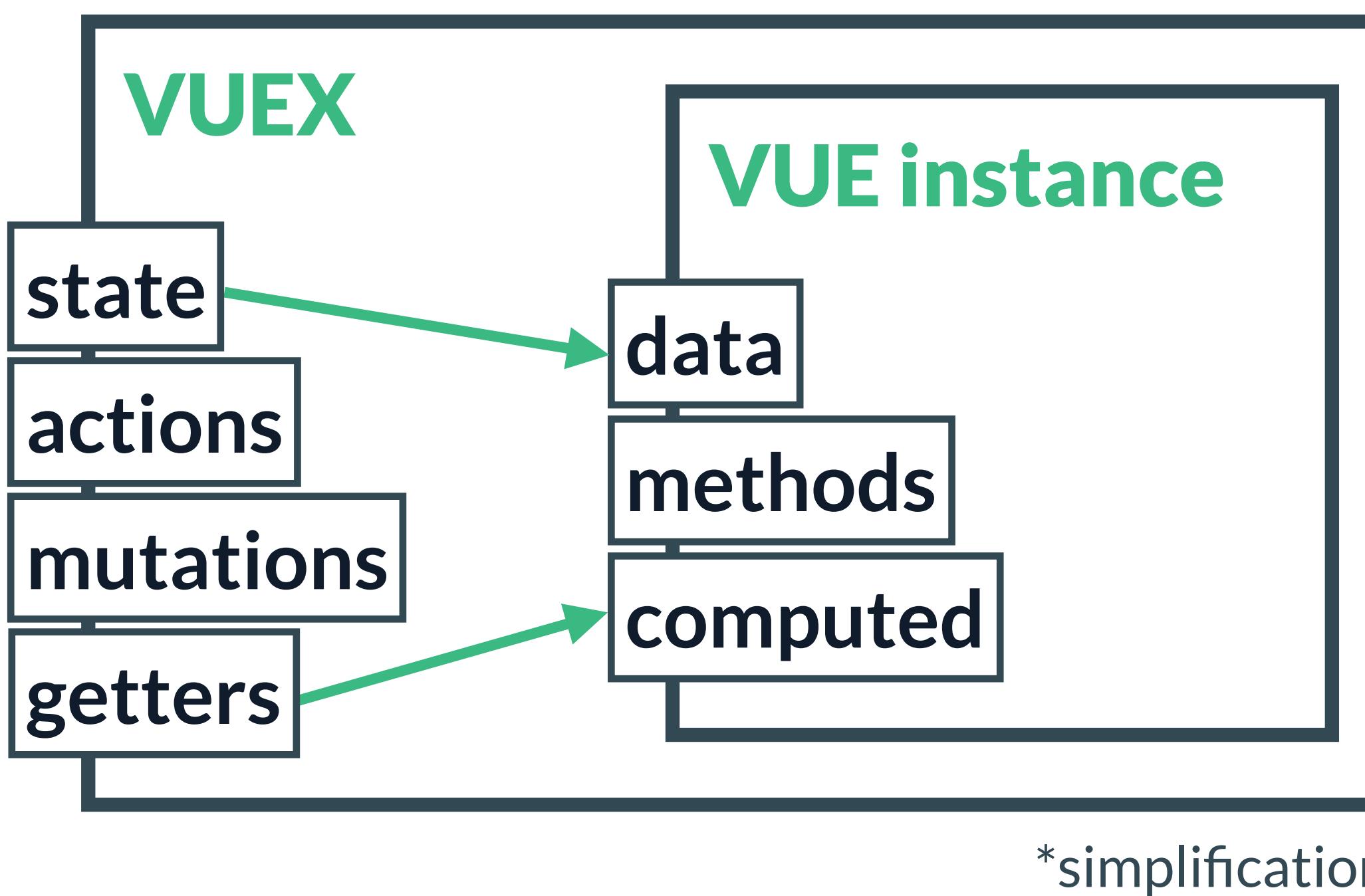
# Ever heard of Vuex?



# Ever heard of Vuex?



# Ever heard of Vuex?



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

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

# 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
}
```



**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

# Composition API solves all those problems

# Composition API

Evolution  
for Applications

# Composition API

REvolution  
for Libraries



# Composition API

You can focus on *what* you want to build rather than on the *how* to do it.

# Composition API

**Learn more**

<https://vue-composition-api-rfc.netlify.com/>

**Don't believe me yet?**

**Don't believe me yet?  
Try it yourself!**

Don't believe me yet?

Try it yourself!

```
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
```



**Still don't believe me?**

# Vuelidate v2.0 alpha

[vuelidate/vuelidate/tree/next](https://github.com/vuelidate/vuelidate/tree/next)

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

# Vuelidate v2.0

## Creating a synchronous validation result

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

# Vuelidate v2.0

## 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
}
```

# 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'

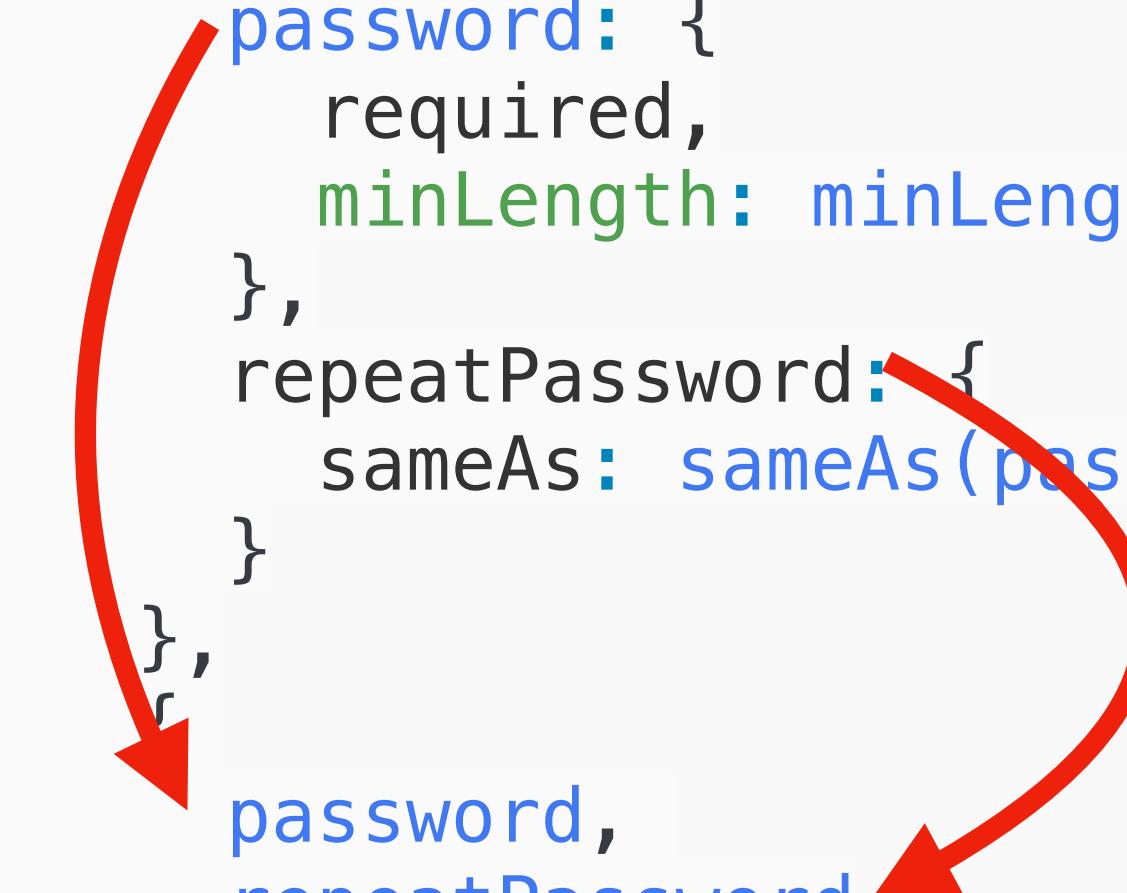
export default {
  setup () {
    const password = ref('')
    const repeatPassword = ref('')
    const $v = useVuelidate(
      rules,
      state
    )
    return {
      password,
      repeatPassword,
      $v
    }
  }
}
```

# 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?

Old API doesn't change.

```
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

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`

# Vuelidate v2.0

## `@vuelidate/validators`

- ❖ Fully tree-shakeable.
- ❖ Separate package.
- ❖ Separate release cycle.
- ❖ Easy to contribute.

# 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 an *accessibility* 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

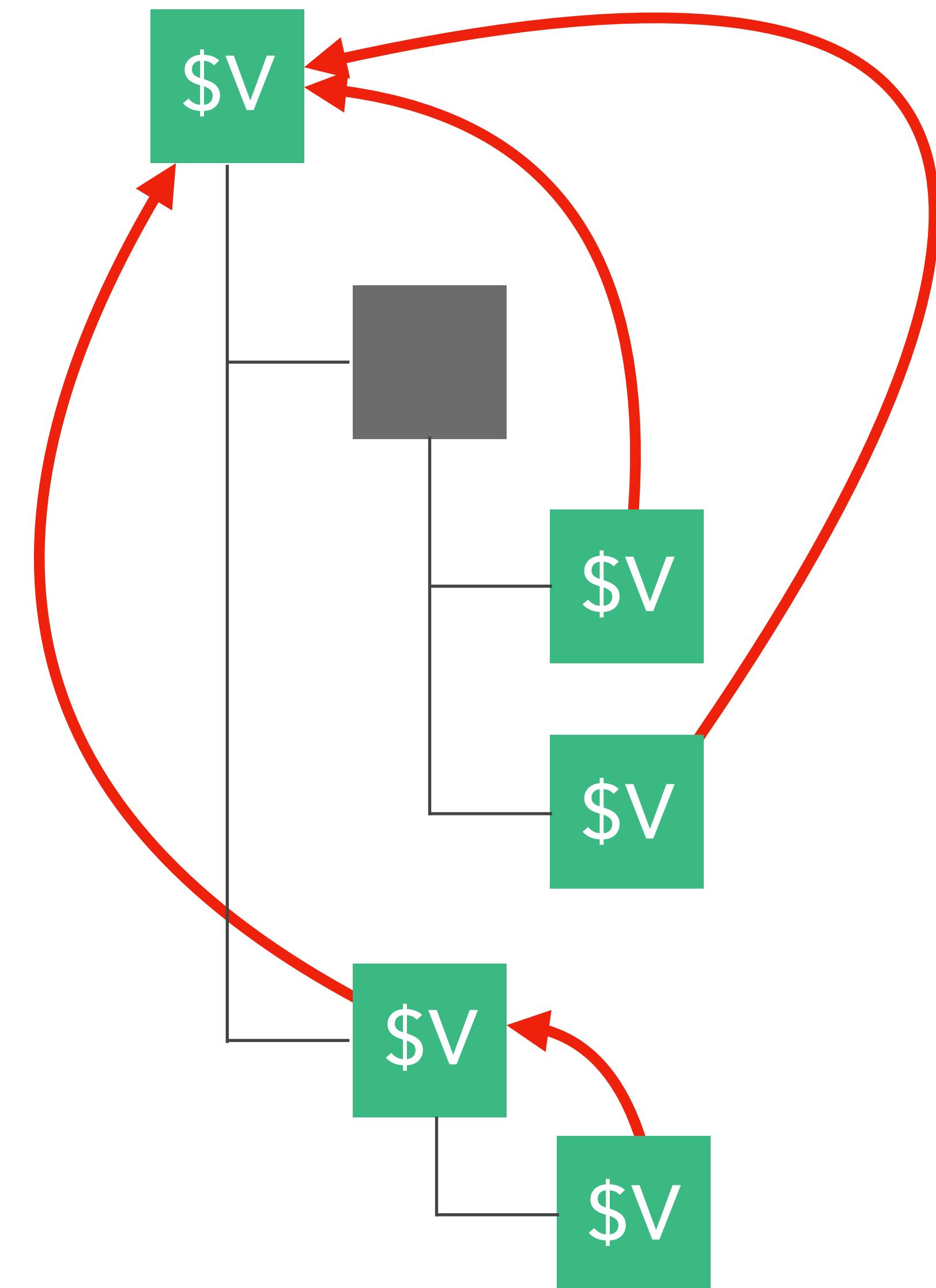
Or move the validators to  
the child component

# Vuelidate v2.0

## 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.

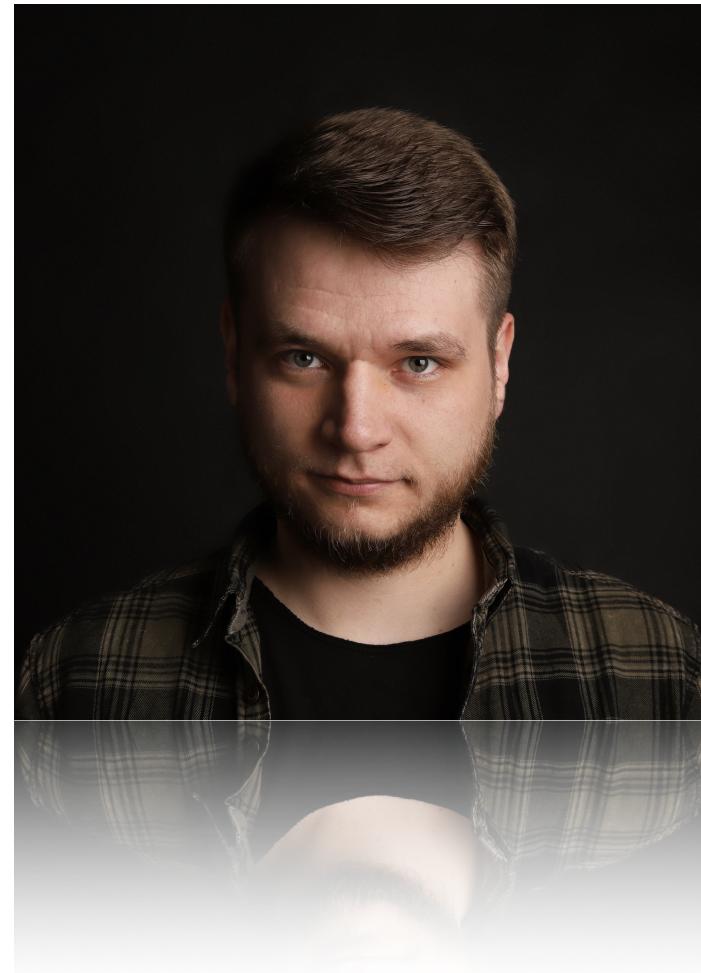


# Vuelidate v2.0

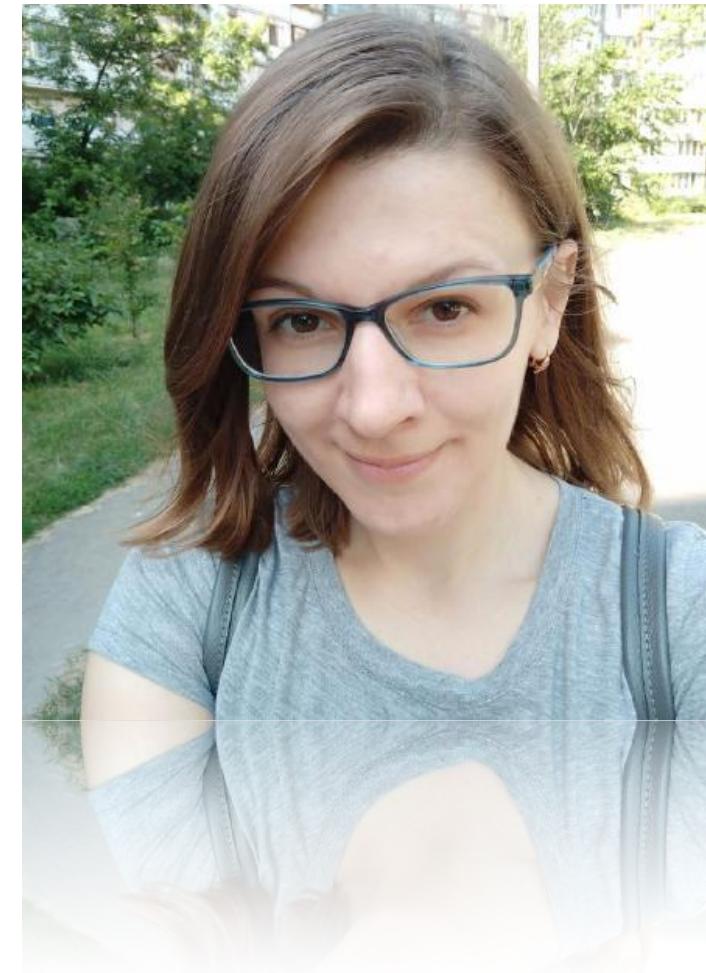
## Roadmap:

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

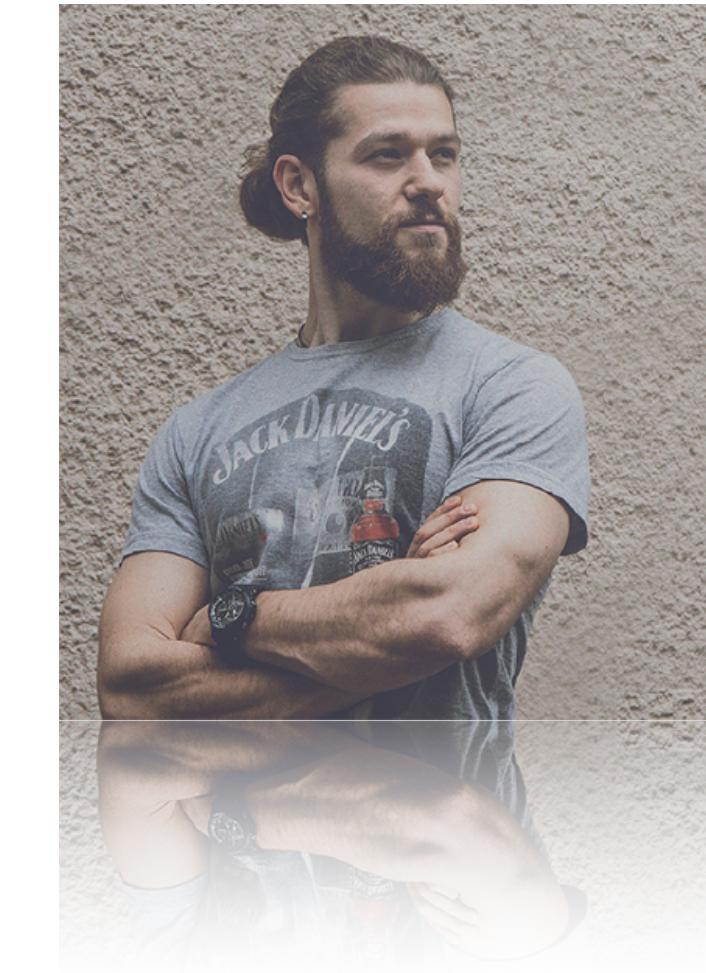
# Meet the Vuelidate team!



That guy on the stage



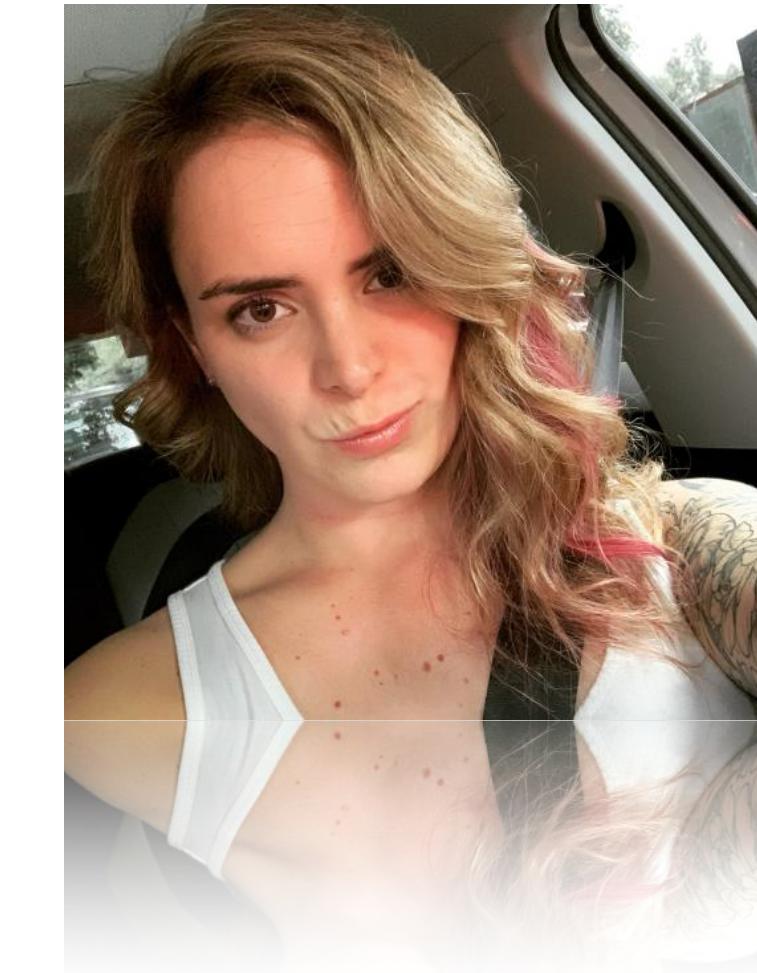
**Natalia Tepluhina**  
[@n\\_tepluhina](https://twitter.com/n_tepluhina)



**Dobromir Hristov**  
[@d\\_m\\_hristov](https://twitter.com/d_m_hristov)



**Ben Hong**  
[@bencodezen](https://twitter.com/bencodezen)



**Marina Mosti**  
[@marinamosti](https://twitter.com/marinamosti)

# One more thing...



# FORMVUELATTE

Lightweight Schema Generated Forms

by Marina Mosti

[vuelidate/formvuelatte](https://github.com/vuelidate/formvuelatte)

# FORMVUELATTE

Lightweight Schema Generated Forms

- ❖ “Bring Your Own Components”
- ❖ Less than 100 LoC
- ❖ Already works with Vue 2.x

Roadmap:

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

**When Vue 3.0 will be ready?**

**When it's ready.**

**Most likely end of Q1, 2020.**

# 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
  }
]
```