Vue.js form validation with vee-validate v4

Abdelrahman Awad

@logaretm

About me

Senior Frontend Engineer at Octopods

I Created and maintain vee-validate  for Vue.js since 2016

I write about Vue.js, JavaScript and TypeScript at my blog logaretm.com

@logaretm

Forms are hard

What makes forms hard?

Value tracking

Validation and error messages

UI/UX

Organization

Debugging DX

About vee-validate

Most popular Vue.js form validation library with

1.4m/month 📈 downloads

8.6k 🌟 on GitHub

Goals

Solve the main pain points:

- Value tracking
- Validation and error messages
- Handle the “80%” of common UI/UX/DX requirements
- Debugging Tools

Offer a progressive API to match Vue.js Ideology

Composition-first design approach

But first...

The building blocks

```
<!-- Native HTML -->
<form>
  <input name="field">
</form>
```

But first...

The building blocks

```
<!-- vee-validate Components -->
<Form>
  <Field name="field" />
</Form>
```

Value tracking

Value tracking

State Binding

State declaration

Value retrieval

```
<template>
  <form>
    <label for="name">Name</label>
    <input id="name" type="text" v-model="name">

    <label for="email">Email</label>
    <input id="email" type="email" v-model="email">

    <label for="password">Password</label>
    <input id="password" type="password" v-model="password">
  </form>
</template>

<script>
export default {
  data: () => ({
    name: '',
    email: '',
    password: '',
  }),
  methods: {
    onSubmit() {
      sendToApi({
        name: this.name,
        email: this.email,
        password: this.password
      })
    }
  }
};
</script>
```

@logaretm

Value tracking: Revised

Implicit model declaration and binding

Auto-retrieval of state

```
{
  name: 'dev',
  email: 'dev@test.com',
  password: '12345',
  links: [
    "https://twitter.com",
    "https://google.com"
  ]
}
```

```
<template>
  <Form @submit="onSubmit">
    <label for="name">Name</label>
    <Field id="name" name="name" type="text">

    <label for="email">Email</label>
    <Field id="email" name="email" type="email">

    <label for="password">Password</label>
    <Field id="password" name="password" type="password">

    <p>Links</p>
    <Field id="links_0" name="links[0]" />
    <Field id="links_1" name="links[1]" />

    <button type="submit">Submit</button>
  </Form>
</template>

<script>
import { Form, Field } from 'vee-validate';

export default {
  components: {
    Form,
    Field
  },
  methods: {
    onSubmit(values) {
      sendToApi(values);
    }
  }
};
</script>
```

Validation

Validation

Multiple ways to validate

- JavaScript Functions and libraries (e.g: validator.js)
- Backend Inspired string expressions (e.g: Laravel's validation syntax)
- Schema validators (e.g: Yup and Zod)

Validation

With JavaScript functions

```
<template>
  <Form @submit="onSubmit">
    <label for="email">Email</label>
    <Field id="email" type="email" :rules="validateField" />
    <ErrorMessage name="email" />

    <button type="submit">Submit</button>
  </Form>
</template>

<script setup>
import { Form, Field, ErrorMessage } from 'vee-validate';
import { isEmail } from 'validator.js';

const validateField = (value) => {
  if (!value) {
    return 'Field is required';
  }

  return isEmail(value) || 'Must be a valid email';
}
</script>
```

Field validation

With Laravel’s string rule expression

```
<template>
 <Form>
    <label for="email">Email</label>
    <Field name="email" type="email" rules="required|email">
    <ErrorMessage name="email" />
  </Form>
</template>

<script setup>
import { Field, ErrorMessage, Form, defineRule } from 'vee-validate';
import { required, email } from '@vee-validate/rules';

defineRule('required', required);
defineRule('email', email);
</script>
```

Field validation

With yup schema

```
<template>
  <Form @submit="onSubmit">
    <label for="email">Email</label>
    <Field id="email" type="email" :rules="schema" />
    <ErrorMessage name="email" />

    <button type="submit">Submit</button>
  </Form>
</template>

<script setup>
import { Form, Field, ErrorMessage } from 'vee-validate';
import { string } from 'yup';

const schema = string().email().required();
</script>
```

Form-level Validation

One schema to rule them all

```
<template>
 <Form :validation-schema="schema">
    <label for="name">Name</label>
    <Field name="name" type="name">
    <ErrorMessage name="name" />

    <label for="email">Email</label>
    <Field name="email" type="email">
    <ErrorMessage name="email" />

    <label for="password">Password</label>
    <Field name="password" type="password">
    <ErrorMessage name="password" />
  </Form>
</template>

<script setup>
import { Field, ErrorMessage, Form } from 'vee-validate';
import { object, string } from 'yup';

const schema = object({
  email: string().email().required(),
  name: string().required(),
  password: string().min(6).required(),
})
</script>
```

@logaretm

Progressive Integration

Progressive Integration

Minimum effort to validate fields

```
<form method="post" action="/register">
  <input placeholder="Name" type="text" name="name">
  <input placeholder="Email" type="email" name="email">
  <input placeholder="Password" type="password" name="password">

  <button>Submit</button>
</form>
```

Progressive Enhancement

```
<Form method="post" action="/register" :validation-schema="schema">
  <Field placeholder="Name" type="text" name="name">
  <ErrorMessage name="name" />

  <Field placeholder="Email" type="email" name="email">
  <ErrorMessage name="email" />

  <Field placeholder="Password" type="password" name="password">
  <ErrorMessage name="password" />

  <button>Submit</button>
</Form>
```

UI / UX

UI / UX

Form interaction flags

```
<template>
<Form v-slot="{ meta, isSubmitting }">
  <!-- Fields -->
  <button :disabled="!meta.valid">
    {{ isSubmitting ? 'Submitting...' : 'Submit '}}
  </button>
</Form>
</template>

<script setup>
import { Form } from 'vee-validate';
</script>
```

UI / UX

Handle Invalid Submissions

```vue
<template>
<Form @invalid-submit="onInvalidSubmit">
  <!-- Fields -->
</Form>
</template>

<script setup>
import { Form } from 'vee-validate';

function onInvalidSubmit({ errors }) {
  const fieldName = Object.keys(errors)[0];
  const el = document.querySelector(`input[name="${fieldName}"]`);
  el?.scrollIntoView();
}
</script>
```

Form Generators

Form Generators

- vee-validate doesn’t offer form generator out of the box but you have all the tools to build your own.
- Full tutorial covers select inputs and custom components, read more here.
- You can use specialized libraries that have first-party integration with vee-validate like Formvuelate.

```vue
<template>
  <Form>
    <div
      v-for="field in fields" :key="field.name">
      <label :for="field.name">{{ field.label }}</label>
      <Field :id="field.name" v-bind="field" />
      <ErrorMessage :name="field.name" />
    </div>

    <button>Submit</button>
  </Form>
</template>

<script setup>
import { Form, Field, ErrorMessage } from 'vee-validate';
import { string } from 'yup';

 const fields =   [
  {
    label: 'Your Name',
    name: 'name',
    as: 'input',
    rules: string().required(),
  },
  {
    label: 'Your Email',
    name: 'email',
    as: 'input',
    rules: string().email().required(),
  },
  {
    label: 'Your Password',
    name: 'password',
    as: 'input',
    type: 'password',
    rules: string().min(6).required(),
  },
];
</script>
```

Composition API

Composition API

- All vee-validate components have composition variants (flavor) of them.
- Think of flavors as outlets to the same API.
- To build field components you use `useField`.
- To build form components you use `useForm`.
- vee-validate components internally use the composition API.

Composition API

Custom field components

```
<template>
<div>
  <label :for="name">{{ label }}</label>
  <input :name="name" :id="name" v-model="value">
  <p>{{ errorMessage }}</p>
</div>
</template>

<script setup>
import { useField } from 'vee-validate';

const props = defineProps({
  name: String,
  label: String
});

const { value, errorMessage } = useField(props.name);
</script>
```

Composition API

Custom form components

```vue
<template>
  <form @submit="onSubmit">
    <!-- Fields -->
  </form>
</template>

<script setup>
import { useForm } from 'vee-validate';

const schema = object({
  name: string().required(),
  email: string().email().required(),
  password: string().min(6).required(),
});

const { handleSubmit } = useForm({
  validationSchema: schema
});

const onSubmit = handleSubmit((values) => {
  console.log(values);
});
</script>
```

Composition API

Before

```vue
<template>
 <Form :validation-schema="schema">
    <label for="name">Name</label>
    <Field name="name" type="name">
    <ErrorMessage name="name" />

    <label for="email">Email</label>
    <Field name="email" type="email">
    <ErrorMessage name="email" />

    <label for="password">Password</label>
    <Field name="password" type="password">
    <ErrorMessage name="password" />
  </Form>
</template>

<script setup>
import { Field, ErrorMessage, Form } from 'vee-validate';
import { object, string } from 'yup';

const schema = object({
  email: string().email().required(),
  name: string().required(),
  password: string().min(6).required(),
})
</script>
```

After

```vue
<template>
  <Form :validation-schema="schema">
    <InputText name="name" label="Name" />
    <InputText type="email" name="email" label="Email Address" />
    <InputText type="password" name="password" label="Password" />

    <button>Submit</button>
  </Form>
</template>

<script setup>
import { Form } from 'vee-validate';
import { object, string } from 'yup';
import InputText from '@/components/InputText.vue';

const schema = object({
  name: string().required(),
  email: string().email().required(),
  password: string().min(6).required(),
});
</script>
```

Vue.js Devtools Plugin

Open this URL and check your Vue.js Devtools tab

https://8k9gy.csb.app/

Sandbox Link

Wait, there is more...

- Array fields
- Multi-step Forms
- Localization
- UI Libraries' integrations

Check the documentation for more information

Thank you 👋

@logaretm