Vuelidate

Validations beyond forms

Главное отличие от всех остальных библиотек - валидация полностью отделена от темплейта.

Для каждого значения, которое нужно валидировать нужно создать ключ внутри опции validations().

```
import Vue from 'vue'
import Validations from 'vuelidate'
import { email, required } from 'vuelidate/lib/validators'

Vue.use(Validations)

new Vue({
 el: '#app',
 data () {
  return {
   email: '',
  }
 },
 validations: {
  email: { required, email } // rules object
  }
}
```

Reserved keywords

Модель $v представляет текущее состояние проверки. Это делается путем свойств, которые содержат выходные данные функций проверки, определенных пользователем.

1. $invalid - показывает состояние модели. True - когда один из валидаторов принимает значение false;
2. $dirty - изменение модели;
3. $error - флаг для показа сообщения. $invalid && dirty;
4. $pending - показывает true, если один из «детей» ожидает асинхронной проверки. Всегда false, если все валидаторы синхронные;
5. $params - кастомные параметры, удобно (нет), например, для добавления кастомных ошибок для валидатора;
6. $each - сохраняет все ключи исходной модели, можно безопасно использовать в циклах.

Есть зарезервированный набор стандартных методов для контроля модели валидации. Работают в связке с обработчиками событий (бинды @input и @blur):

1. $touch - устанавливает dirty flag;
2. $reset - сбрасывает валидацию;
3. $flattenParams - получает массив параметров проверки для всех валидаторов, существующих в этом объекте проверки.

Usage

1. Вложенные данные. Можно проверять настолько глубоко, насколько это возможно.

```
export default {
  data () {
    return {
      form: {
        nestedA: '',
        nestedB: ''
      }
    }
  },
  validations: {
    form: {
      nestedA: {
        required
      },
      nestedB: {
        required
      }
    }
  }
}
```

2. Группы валидаций. Если хотим создать валидатор, который объединяет в себя несколько:
validationGroup: [‘form.nestedA’, ‘form.nestedB’]

Usage

3. Асинхронные валидаторы.
Поддержка async предоставляется из коробки, нужно просто использовать валидатор который возвращает промис. Значение промиса используется для проверки напрямую. Доступ к данным любого компонента должен осуществляться синхронно для правильного реактивного поведения. Нужно сохранить его как переменную в области проверки, если вам нужно использовать его в любом асинхронном обратном вызове, например, в .then.

```
<div>
  <div class="form-group" v-bind:class="{ 'form-group--error': $v.username.$error, 'form-group--loading': $v.username.$pending }">
    <input class="form__input" v-model.trim="username" @input="$v.username.$touch()">
  </div><span class="form-group__message" v-if="!$v.username.required">Username is required.</span><span class="form-group__message"
v-if="!$v.username.isUnique">This username is already registered.</span>
</div>
export default {
  data () {
    return {
      username: ''
    }
  },
  validations: {
    username: {
      required,
      isUnique (value) {
        if (value === '') return true
        return new Promise((resolve, reject) => {
          setTimeout(() => {
            resolve(typeof value === 'string' && value.length % 2 !== 0)
          }, 350 + Math.random() * 300)
        })
      }
    }
  }
}
```

Usage

4. Dynamic parameters.

```
export default {
  data () {
    return {
      name: '',
      minLength: 3,
      valName: 'validatorName'
    }
  },
  validations () {
    return {
      name: {
        [this.valName]: minLength(this.minLength)
      }
    }
}
```