

Forms



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://about.softuni.bg>

Table of Contents

1. Two-way data binding
2. Working with forms
3. Validating forms with Vuelidate



Have a Question?



sli.do

#vue-js



Two-way Data Binding

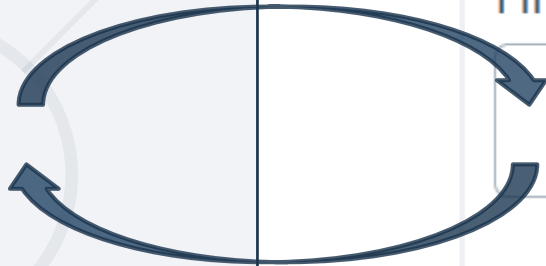
What is "two-way" data binding?

- Feature that allows automatic synchronization of data between a user interface element and the dedicated state variable
- When you update the UI element, it updates the data, and when you update the data, it updates the UI element

```
<script>  
  export default {  
    data() {  
      return {  
        name: []  
      }  
    }  
  }  
</script>
```

First name

First name



- Achieved using the **v-model** directive
 - Essentially syntax sugar for updating data
 - Based on user input events
 - Supports input modifiers
- Automatically picks the correct way to update an element based on the input type

```
<input  
  :value="text"  
  @input="event => text = event.target.value">
```

```
// With v-model we can simplify it to  
<input v-model="text">
```

- `<input>` with text types and `<textarea>` elements use **value** property and **input** event
- `<input type=checkbox>` and `<input type=radio>` use **checked** property and **change** event.
- `<select>` use **value** as a prop and **change** as an event.

```
<input v-model="text">  
<textarea v-model="message"></textarea>
```

```
<label for="terms">  
  <input type="checkbox" id="terms" name="terms" v-model="checked">  
  I agree to the Terms and Conditions: {{ checked }}  
</label>
```

Multiple checkboxes

```
<template>
  <div>
    <div>Checked names: {{ checkedNames
  }}</div>

    <input type="checkbox" id="jack"
value="Jack" v-model="checkedNames">
    <label for="jack">Jack</label>

    <input type="checkbox" id="john"
value="John" v-model="checkedNames">
    <label for="john">John</label>

    <input type="checkbox" id="mike"
value="Mike" v-model="checkedNames">
    <label for="mike">Mike</label>
  </div>
</template>
```

```
<script>
  export default {
    data() {
      return {
        checkedNames: []
      }
    }
  }
</script>
```

Checked names: ["Mike"]

☐ Jack ☐ John ☒ Mike

Radio button

```
<div>Picked: {{ picked }}</div>
```

```
<input type="radio" id="one" value="One" v-model="picked" />  
<label for="one">One</label>
```

```
<input type="radio" id="two" value="Two" v-model="picked" />  
<label for="two">Two</label>
```

Picked: One

☒ One ☐ Two

```
<div>Selected: {{ selected }}</div>
```

```
<select v-model="selected">  
  <option disabled value="">Please select one</option>  
  <option>A</option>  
  <option>B</option>  
  <option>C</option>  
</select>
```

Selected: A

A ▼

- For **radio**, **checkbox** and **select** options, the **v-model** binding values are usually static strings

```
<!-- `toggle` is either true or false -->
<input type="checkbox" v-model="toggle"
/>
```

```
<!-- `picked` is a string "a" when
checked -->
<input type="radio" v-model="picked"
value="a" />
```

```
<!-- `selected` is a string "abc" when
the first option is selected -->
<select v-model="selected">
  <option value="abc">ABC</option>
</select>
```

```
// Checkbox
<input
  type="checkbox"
  v-model="toggle"
  true-value="yes"
  false-value="no" />
```

```
// Radio
<input type="radio" v-model="pick" :value="first"
/>
<input type="radio" v-model="pick" :value="second"
/>
```

```
<select v-model="selected">
  <option :value="{ number: 123 }">123</option>
</select>
```

- **.lazy**

- Change v-model to syncs the input with the data after change event instead after each input event

```
<!-- synced after "change" instead of "input" -->  
<input v-model.lazy="msg" />
```

- **.number**

- Automatically typecast as number
- Applied automatically if `<input type="number"/>`

```
<input v-model.number="age" />
```

- **.trim**

- automatically trim whitespace

```
<input v-model.trim="msg" />
```

V-model on Vue component

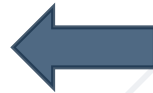
- `<CustomInput>` component must do two things:
- Bind the value attribute of a native `<input>` element to the **modelValue** prop
- When a native input event is triggered, emit an **update:modelValue** custom event with the new value

```
<input
  :value="searchText"
  @input="searchText =
    $event.target.value"
/>
```



```
<script>
export default {
  props: ['modelValue'],
  emits: ['update:modelValue']
}
</script>

<template>
  <input
    :value="modelValue"
    @input="$emit('update:modelValue',
      $event.target.value)"
  />
</template>
```



```
<CustomInput v-model="searchText" />
```

Multiple v-model bindings

- v-model on a component uses modelValue as the prop and update:modelValue as the event
- We can modify these names passing an argument to v-model:

```
<MyComponent  
  v-model:title="bookTitle"  
>
```

```
<UserName  
  v-model:first-name="first"  
  v-model:last-name="last"  
>
```



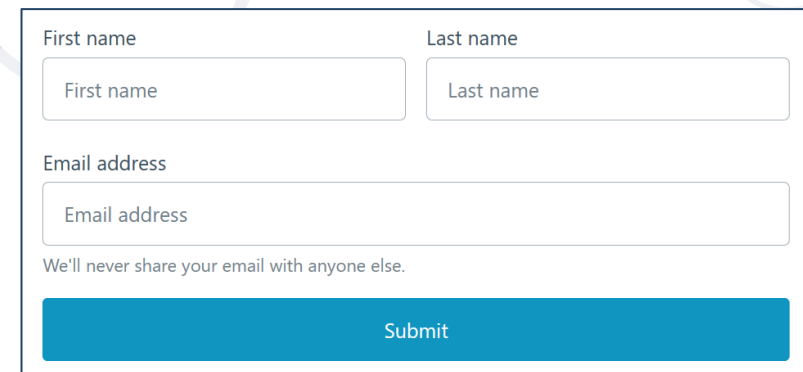
```
<script>  
export default {  
  props: {  
    firstName: String,  
    lastName: String  
  },  
  emits: ['update:firstName',  
    'update:lastName']  
}  
</script>  
  
<template>  
  <input  
    type="text"  
    :value="firstName"  
    @input="$emit('update:firstName',  
      $event.target.value)"  
  />  
  // Second input using lastName  
</template>
```



Forms

- **Forms help us with:**
 - Collecting user input
 - Validating data
 - Enabling user interaction
 - Structured data submission
- **Examples of forms include:**
 - Registering or logging in
 - Submitting help requests
 - Placing orders
 - Booking flights, and more

- Create a registration form with the following input fields
 - First name (text)
 - Last name (text)
 - Email (text)
 - Age (number)
 - Skill Set (checkbox)
 - Gender (radio)
 - Country (select)
- Use v-model on all input fields
- Handle the submit event and prevent the default behavior
- After submitting -> show the data you have entered
- Bonus: Create a reusable input field component for the text fields



First name Last name

First name Last name

Email address

Email address

We'll never share your email with anyone else.

Submit



Validation

Using the Vuelidate library

- **Vuelidate - A Simple, Lightweight Model-Based Validation Plugin**
 - **Flexibility:** Vuelidate offers flexibility and decouples validation logic from templates.
 - **Lightweight:** It's dependency-free and a minimalistic library.
 - **Built-In Validators:** Vuelidate provides numerous built-in validators for common use cases.
 - **Custom Validators:** It's easy to use with custom validators.
 - **Documentation:** For more details, you can refer to the [official documentation](#).

Installation and usage

1. Install the library with npm

```
npm install @vuelidate/core  
@vuelidate/validators
```

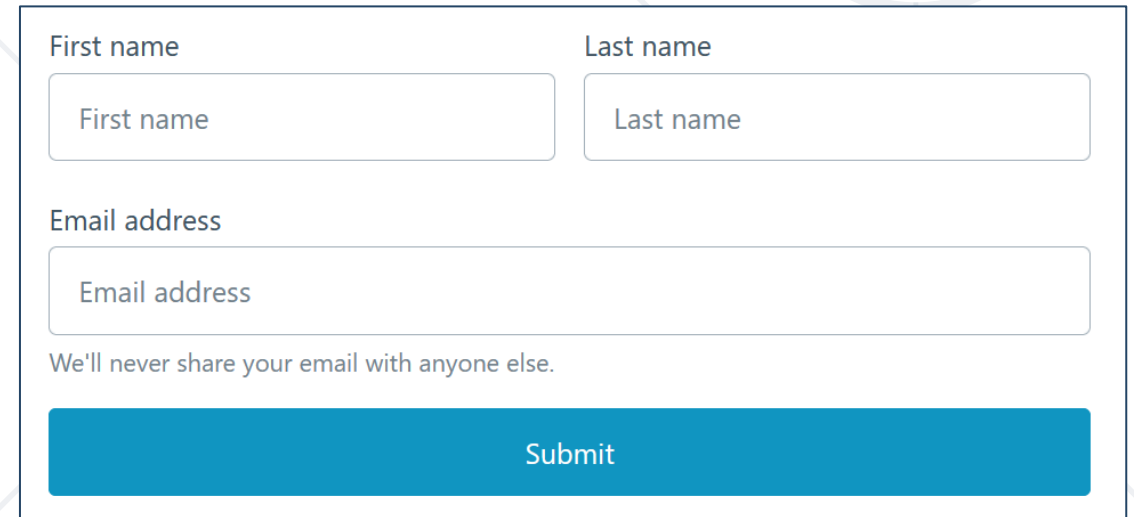
2. Import the **useVuelidate** hook and desired validations in your component

```
import { useVuelidate } from  
'@vuelidate/core'  
import { required, email } from  
'@vuelidate/validators'
```

3. Inside your Options API structure create a **setup** option, that returns our initialized hook and a new **validations** option, where you describe your desired data and structure to be validated

```
export default {  
  setup () {  
    return { v$: useVuelidate() }  
  },  
  data () {  
    return {  
      firstName: '',  
      lastName: '',  
      contact: {  
        email: ''  
      }  
    }  
  },  
  validations () {  
    return {  
      firstName: { required }, // Matches this.firstName  
      lastName: { required }, // Matches this.lastName  
      contact: {  
        email: { required, email } // Matches  
        this.contact.email  
      }  
    }  
  }  
}
```

- Take the form from your previous exercise
- Install Vuelidate and connect it to your form component
- Add validation with Vuelidate to each field
 - First name - required
 - Last name - required
 - Email – required & email
 - Age – min 1 and max 100
 - Skill Set - required
 - Gender - required
 - Country - required



First name

Last name

Email address

We'll never share your email with anyone else.

Submit



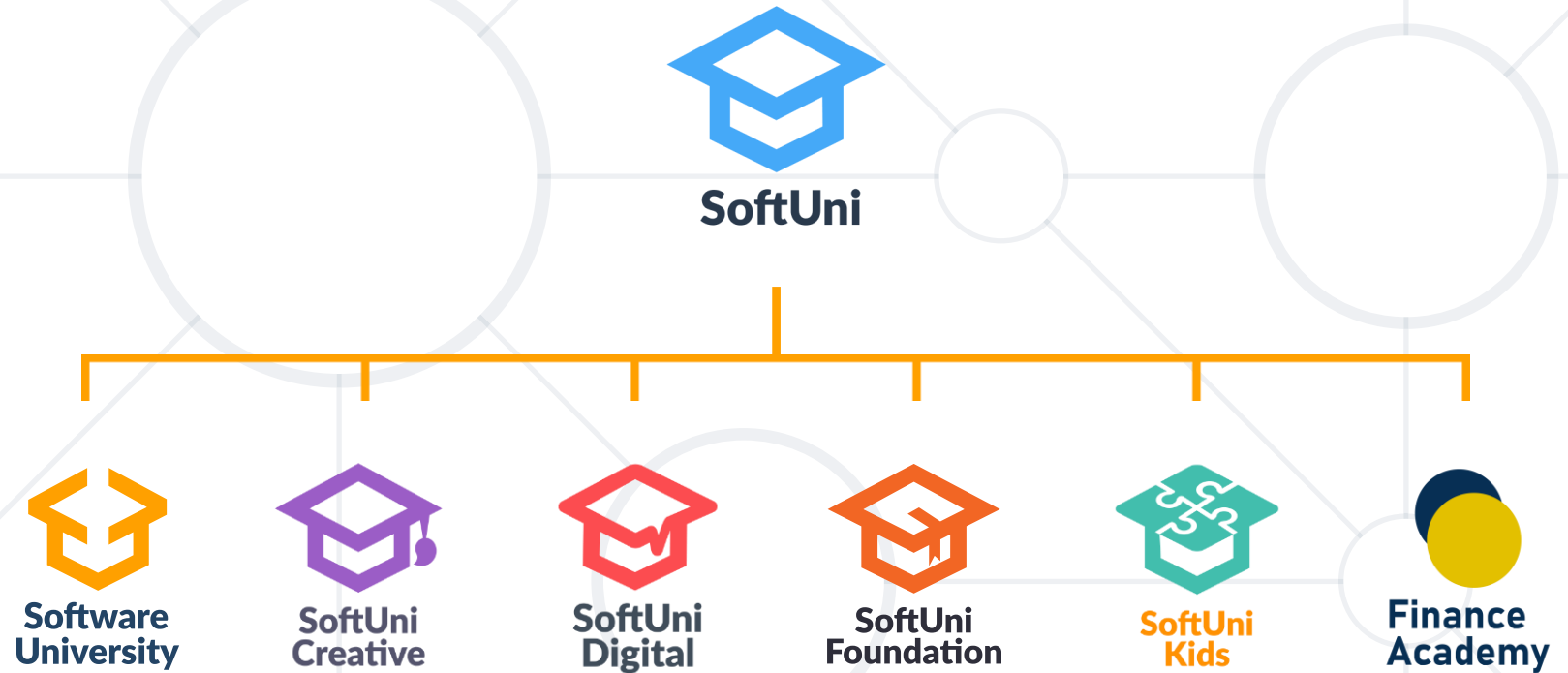
Practice

Live Exercise in Class (Lab)

- V-model helps us work faster and easier with our form elements
- V-model is customizable and can be used on custom Vue components too
- Vuelidate helps us validate our forms before submitting to the server



Questions?



SoftUni Diamond Partners

**SUPER
HOSTING
.BG**



**Coca-Cola HBC
Bulgaria**

 **Flutter**TM
International

INDEAVR
Serving the high achievers



AMBITIONED

 **DRAFT
KINGS**



BOSCH

 **Postbank**
Решения за твоето утре

 **PHAR
VISION**



SmartIT

DXC
TECHNOLOGY

createX

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg

