Handling User Input with Forms

A Basic `<input>` Form Binding

We can use **v-model** directive for two-way binding data properties with form's input.

Grouping Data and Pre-populating Input

Instead of define multiple data properties separately. We can group them into **javascript object** property.

```
// App.vue
<template>
...
    <div class="form-group">
        <label for="email">Mail</label>
        <input
                type="text"
                id="email"
                class="form-control"
                v-model.lazy="userData.email">
    </div>
    {{ userData.email }}
    <div class="form-group">
        <label for="password">Password</label>
        <input
                type="password"
                id="password"
                class="form-control"
                v-model="userData.password">
    </div>
    <div class="form-group">
        <label for="age">Age</label>
        <input
```

```
                   type="number"
                   id="age"
                   class="form-control"
                   v-model="userData.age">
    </div>
...
</template>

<script>
export default {
  // data: function () {...}
  data() {
      return {
          // an object of user data
          userData: {
              email: '',
              password: '',
              age: 28
          }
      }
    }
}
</script>
```

Modifying User Input with Input Modifier

There are many **v-model modifiers** that give different behaviors we can use.

- v-model: default behavior (fire every keystroke)
    - suggestion and real-time validation
- v-model.lazy: fire only when lose focus
    - for submit or validation
- v-model.trim: get rid of white spaces
- v-model.number: convert input to number
- v-model.lazy.trim.number

Binding <textarea> and Saving Line Breaks

By default textarea store whire space and line breaks. It just does not show in the display area. The only thing you need to do is to add a proper **style** at the out.

```
<template>
...
  <p>Mail: {{ userData.email }}</p>
  <p>Password: {{ userData.password }}</p>
  <p>Age:{{ userData.age }}</p>

  <!-- add proper style here!!! -->
  <p style="white-space: pre">Message: {{ message }}
</p>

...
</template>
```

Using Checkboxes and Saving Data in Array

We can use an **array** to stores multiple selections. **v-model** does many tricks to help binding array with selections.

```
<template>
...
  <div class="form-group">
      <label for="sendmail">
          <input
              type="checkbox"
              id="sendmail"
              value="SendMail"
              v-model="sendMail"> Send Mail
      </label>
      <label for="sendInfomail">
```

```
          <input
              type="checkbox"
              id="sendInfomail"
              value="SendInfoMail"
              v-model="sendMail"> Send Infomail
      </label>
  </div>
...
  <p><strong>Send Mail?</strong></p>
  <ul>
      <li v-for="item in sendMail" :key="item">{{
item }}</li>
  </ul>
...
</template>

<script>
...
  data() {
    return {
        ...
        sendMail: [],
        ...
    }
  }
...
</script>
```

Using Radio Buttons

```
<template>
...
  <label for="male">
  <input
      type="radio"
```

```
        id="male"
        value="Male"
        v-model="gender"> Male
  </label>
  <label for="female">
  <input
        type="radio"
        id="female"
        value="Female"
        v-model="gender"> Female
  </label>
...
  <p>Gender: {{ gender }}</p>
...
</template>

<script>
...
  data() {
    return {
        ...
        gender: 'Male',
        ...
    }
  }
...
</script>
```

Handling Dropdowns with <select> and <option>

There are two parts. Firstly, we can define the list of options as **array** in data property and use them to **populate** the option tag in template. Secondly, we need to define another data property to store the **selection** from user. We are able to set the default selection as well.

```
<template>
...
  <label for="priority">Priority</label>
  <select
      id="priority"
      class="form-control"
      v-model="selectedPriority">

      <option
        v-for="p in priorities"
        :key="p"
        :selected="p == 'Medium'">{{ p }}</option>
  </select>
...
  <p>Gender: {{ gender }}</p>
...
</template>

<script>
...
  data() {
    return {
        ...
        priorities: ['High','Medium','Low'],
        selectedPriority: 'High'
        ...
    }
  }
...
</script>
```

v-model and Custom Control

Build your own input by create a component. **v-model** does many things behind the scene: bind, listen to event, and update the data.

```
< ... v-model="data" ... >

  is similar to

< ... v-bind:value="data"
      v-on:input="data = $event.target.value" ...>
  OR
< ... :value="data"
      @change="data = $event.target.value" ...>
```

*Note: using *@input* or *@change* depends on **v-model modifier** type.

The input component needs to have the **value** attribute.

The data property is passed from the parent to the child component using **v-model="data"** in the client's placeholder.

```
// App.vue - parent
<template>
...
  <app-switch v-model="dataSwitch"></app-switch>
...
</template>

<script>
...
  data() {
    return {
        ...
        dataSwitch: true,
        ...
    }
  }
...
</script>
```

The child read the passed data via **probs: ['value'].** After the child updates the value, it then passed back the data to its parent using **custom event** that sends **'input'** type and **data** to pass back.

```
// Switch.vue - parent
<template>
  <div>
    <div
      id="on"
      @click="switched(true)"
      :class="{active: value}">On</div>

    <div
      id="off"
      @click="switched(false)"
      :class="{active: !value}">Off</div>

  </div>
</template>

<script>
export default {
  data() {
    return {
      props: ['value']
    }
  },
  methods: {
    switched(isOn) {
      // custom input: v-model is waiting for 'input'
      this.$emit('input', isOn);
    }
  }

}
```

```
</script>
```

Submitting a Form

```
<template>
...
  <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-offset-2
col-md-6 col-md-offset-3">

          <button
              class="btn btn-primary"
              @click.prevent="submitted">Submit!
              <!-- let VueJS handle the submit -->

          </button>
      </div>
  </div>
...
  <div class="row" v-if="isSubmitted">
    <!-- display all information -->
    ...
  </div>

</template>

<script>
...
  data() {
    return {
        ...
        isSubmitted = false;
        ...
    }
  },
  methods: {
    submitted() {
        this.isSubmitted = true;
        // do somethingelse:
        // validation, save, post, ...
    }
  },
...
</script>
```