**Section 8 – Component Communication**

Chapter 89 – Introducing “Props” (Parent => Child Communication)

1. Constructor

<friend-contact></friend-contact>

It can be a constructor to send the parameter, the code will be like below.

<friend-contact

        name = "Manuel Lorenz"

        phone-number = "01234 78992"

        email-address = "manuel@localhost.com"

></friend-contact>

To get the inputted parameter, we add props

export default {

  props: [

    'name',

    'phoneNumber',

    'emailAddress'

  ],

  data() {

    return {

      detailsAreVisible: false,

      friend: {

        id: "manuel",

        name: "Manuel Lorenz",

        phone: "0123 45678 90",

        email: "manuel@localhost.com",

      },

    };

  },

  methods: {

    toggleDetails() {

      this.detailsAreVisible = !this.detailsAreVisible;

    }

  }

};

Chapter 90 – Prop Behavior And Changing Props

1. Note: If you want to add a new variable, you need to add from the parent, like below.

<friend-contact

        name = "Manuel Lorenz"

        phone-number = "01234 78992"

        email-address = "manuel@localhost.com"

        is-favorite = "1"

      ></friend-contact>

The result will be like below.

export default {

  props: [

    'name',

    'phoneNumber',

    'emailAddress',

    'isFavorite'

  ],

  data() {

    return {

      detailsAreVisible: false,

      friend: {

        id: "manuel",

        name: "Manuel Lorenz",

        phone: "0123 45678 90",

        email: "manuel@localhost.com",

      },

      friendIsFavorite: this.isFavorite,

    };

  },

  methods: {

    toggleDetails() {

      this.detailsAreVisible = !this.detailsAreVisible;

    },

    toggleFavorite() {

      if (this.friendIsFavorite === '1') {

        this.friendIsFavorite = '0';

      } else {

        this.friendIsFavorite = '1';

      }

    }

  }

};

Chapter 91 – Validating Props

1. We can make validation for our input through props.

export default {

  // props: ['name', 'phoneNumber', 'emailAddress', 'isFavorite'],

  props: {

    name: {

      type: String,

      required: true,

    },

    phoneNumber: {

      type: String,

      required: true,

    },

    emailAddress: {

      type: String,

      required: true,

    },

    isFavorite: {

      type: String,

      required: false,

      default: '0',

      validator: function(value) {

        return value === '1' || value === '0';

      }

    },

  },

  data() {

    return {

      detailsAreVisible: false,

      friend: {

        id: "manuel",

        name: "Manuel Lorenz",

        phone: "0123 45678 90",

        email: "manuel@localhost.com",

      },

      friendIsFavorite: this.isFavorite,

    };

  },

  methods: {

    toggleDetails() {

      this.detailsAreVisible = !this.detailsAreVisible;

    },

    toggleFavorite() {

      if (this.friendIsFavorite === '1') {

        this.friendIsFavorite = '0';

      } else {

        this.friendIsFavorite = '1';

      }

    }

  }

};

Chapter 92 – Supported Prop Values

In general, you can learn all about prop validation in the official docs: <https://v3.vuejs.org/guide/component-props.html>

Specifically, the following value types (type property) are supported:

* String
* Number
* Boolean
* Array
* Object
* Date
* Function
* Symbol

But type can also be any constructor function (built-in ones like Date or custom ones).

Chapter 93 – Working With Dynamic Prop Values

1. We can use v-for to do looping and get all the inputted data.

<friend-contact

        v-for = "friend in friends"

        :key = "friend.id"

        :name = "friend.name"

        :phone-number = "friend.phone"

        :email-address = "friend.email"

        :is-favorite = "true"

      ></friend-contact>

And the inputted data is below.

<script>

export default {

  data() {

    return {

      friends: [

        {

          id: "manuel",

          name: "Manuel Lorenz",

          phone: "0123 45678 90",

          email: "manuel@localhost.com",

        },

        {

          id: "julie",

          name: "Julie Jones",

          phone: "0987 654421 21",

          email: "julie@localhost.com",

        },

      ],

    };

  },

};

</script>

Chapter 94 – Emitting Custom Events (Child => Parent Communication)

1. We can connect parent’s method with child’s method like below. The child method like below.

methods: {

    toggleDetails() {

      this.detailsAreVisible = !this.detailsAreVisible;

    },

    toggleFavorite() {

      this.$emit('toggle-favorite', this.id);

    }

  }

And the parent’s method like below.

methods: {

    toggleFavoriteStatus(friendId) {

      const identifiedFriend = this.friends.find(

        (friend) => friend.id === friendId

      );

      identifiedFriend.isFavorite = !identifiedFriend.isFavorite;

    }

  }

Chapter 95 – Defining And Validating Custom Events

1. We can do some validation through emits in our children, the code like below.

emits: {

    'toggle-favorite': function(id) {

      if (id) {

        return true;

      } else {

        console.warn('ID is missing!');

        return false;

      }

    }

  },

Chapter 96 – Prop / Event Fall through And Binding All Props

1. There are two advanced concepts you also should have heard about:

* Prop Fallthrough
* Binding All Props on a Component

1. Prop Fallthrough

You can set props (and listen to events) on a component which you haven't registered inside of that component. For example:

**BaseButton.vue**

<template>

<button>

<slot></slot>

</button>

</template>

<script>export default {}</script>

This button component (which might exist to set up a button with some default styling) **has** **no special props** that would be registered.

Yet, you can use it like this:

<base-button type="submit" @click="doSomething">Click me</base-button>

Neither the type prop nor a custom click event are defined or used in the BaseButton component.

**Yet, this code would work.**

Because Vue has built-in support for **prop (and event) "fallthrough"**.

Props and events added on a custom component tag **automatically fall through** to the **root component** in the template of that component. In the above example, type and @click get added to the <button> in the BaseButton component.

You can get access to these fallthrough props on a built-in $attrs property (e.g. this.$attrs).

This can be handy to build "utility" or pure presentational components where you don't want to define all props and events individually.

**You'll see this in action** the component course project ("Learning Resources App") later.

You can learn more about this behavior here: <https://v3.vuejs.org/guide/component-attrs.html>

#### Binding all Props

There is another built-in feature/ behavior related to props.

If you have this component:

**UserData.vue**

<template>

<h2>{‌{ firstname }} {‌{ lastname }}</h2>

</template>

<script>

export default {

props: ['firstname', 'lastname']

}

</script>

You **could** use it like this:

<template>

<user-data :firstname="person.firstname" :lastname="person.lastname"></user-data>

</template>

<script>

export default {

data() {

return {

person: { firstname: 'Max', lastname: 'Schwarz' }

};

}

}

</script>

But if you have an object which holds the props you want to set as properties, you can also **shorten the code a bit**:

<template>

<user-data v-bind="person"></user-data>

</template>

<script>

export default {

data() {

return {

person: { firstname: 'Max', lastname: 'Schwarz' }

};

}

}

</script>

With v-bind="person" you pass all key-value pairs inside of person as props to the component. That of course requires person to be a JavaScript object.

This is purely optional but it's a little convenience feature that could be helpful.

Chapter 97 – Demo: Adding Components And Connecting Them

* 1. We try to add one button to Add New Contact. In App.vue the code like below (parent).

methods: {

    addContact(name, phone, email) {

      const newFriendContact = {

        id: new Date().toISOString(),

        name: name,

        phone: phone,

        email: email,

        isFavorite: false

      };

      this.friends.push(newFriendContact);

    }

  }

The child code will be like below.

export default {

    emits: ['add-contact'],

    data() {

        return {

            enteredName: '',

            enteredPhone: '',

            enteredEmail: ''

        };

    },

    methods: {

        submitData() {

            this.$emit('add-contact',

                this.enteredName,

                this.enteredPhone,

                this.enteredEmail

            );

        }

    }

};

And we need to add @submit at the form which want to submit the new contact.

<form @submit.prevent="submitData">

Chapter 98 – Demo: Adding More Component Communication

1. To delete the contact we can use the code below.

deleteContact(friendId) {

      this.friends = this.friends.filter((friend) => friend.id !== friendId);

}

Chapter 99 – A Potential Problem

1. Sending data from parent to child (parent, child, child, and child)

Chapter 100 – Provide + Inject To The Rescue

1. To solve the problem we just use provide like below.

provide: {

    topics: [

        {

          id: 'basics',

          title: 'The Basics',

          description: 'Core Vue basics you have to know',

          fullText:

            'Vue is a great framework and it has a couple of key concepts: Data binding, events, components and reactivity - that should tell you something!',

        },

        {

          id: 'components',

          title: 'Components',

          description:

            'Components are a core concept for building Vue UIs and apps',

          fullText:

            'With components, you can split logic (and markup) into separate building blocks and then combine those building blocks (and re-use them) to build powerful user interfaces.',

        },

      ],

  },

And to get the data from provide we can use inject like below.

<script>

export default {

  inject: ['topics'],

  emits: ['select-topic']

};

</script>

Note: we need a relation between parent and child, so neighbor can not get the data (topics)

1. The problem is we still have 2 arrays for topics, in App.vue (data and provide)

Solution: Write provide like a method like below.

data() {

    return {

      topics: [

        {

          id: 'basics',

          title: 'The Basics',

          description: 'Core Vue basics you have to know',

          fullText:

            'Vue is a great framework and it has a couple of key concepts: Data binding, events, components and reactivity - that should tell you something!',

        },

        {

          id: 'components',

          title: 'Components',

          description:

            'Components are a core concept for building Vue UIs and apps',

          fullText:

            'With components, you can split logic (and markup) into separate building blocks and then combine those building blocks (and re-use them) to build powerful user interfaces.',

        },

      ],

      activeTopic: null,

    };

  },

  provide() {

    return {

      topics: this.topics

    };

  },

1. To add a new topic in several minutes, we can use the code below.

mounted() {

    setTimeout(() => {

      this.topics.push({

        id: 'events',

        title: 'Events',

        description: 'Events are important in Vue',

        fullText: 'Events allow you to trigger code on demand!'

      });

    }, 3000);

  }

Chapter 101 – Provide + Inject For Functions / Methods

1. How to pass a method from parent to children without write it on the tag of component ?

provide() {

    return {

      topics: this.topics,

      selectTopic: this.activateTopic

    };

  },

  methods: {

    activateTopic(topicId) {

      this.activeTopic = this.topics.find((topic) => topic.id === topicId);

    },

  },

In the child vue, we can type like below.

export default {

  inject: ['selectTopic'],

  props: ['id', 'topicName', 'description']

};

And to activate the button, we can write the code like below.

<button @click="selectTopic(id)">Learn More</button>

Chapter 102 – Provide + Inject VS Props And Custom Events

1. It is important to communicate between parent and child. Provide and Inject is a better solution than using v-bind and v-on.

Chapter 103 – Module Summary

