

# Module 3-13

## VUE Nested Components

# Working With Multiple Components

---

- So far we have dealt with a relatively simple project setup, with a single component doing all the hard work, then importing that component into the App.vue parent component.
- The whole idea of component based JS development is that we can take several components and integrate them together under a parent component.
- Namely, we will be covering how to transmit data from a parent component to a child component and vice versa.

# Properties

— — —

- Properties can be thought of as characteristics of the component.
- We will be using properties to transmit data **from a parent component to a child component**.

# Properties Defining

- Within a child component, props are defined in 1 of 2 ways.
- It is a peer of the other sections we've seen so far: data(), computed, methods.

We can also  
define props  
using an array:

```
props: ["rating"],  
data() {...}
```

```
<script>  
export default {  
  name: "product-review",  
  props: {  
    // your props go here  
  },  
  data() {  
    ...  
  },  
  computed: {  
    ...  
  },  
  methods: {  
    ...  
  }  
}  
</script>
```

# Properties Example

Consider the following example, where we utilize props to send data from a parent component to a child component:

## On App.VUE (or parent component)

```
<template>
  <div id="app">
    ...
    <product-review
      name="Toothbrush"
      description="To brush teeth"
    />
    ...
  </div>
</template>
```

## Child Component (script)

```
props: {
  name: String,
  description: String,
},
```

The child component can now use these props, for instance having **{{ name }}** on the template would result in *Toothbrush* being rendered.

**Let's Define Some Props**

# VUEx & State Management

— — —

- We have now seen several situations where the controls on our page are dependant upon the values of a data object.
- Keeping our page elements consistent with our data object is known as **state management**.
- This can get complicated real fast, when multiple components reading from and/or changing the data object.

# VUEx & State Management

— — —

- VUEx helps us practice good state management in the following ways:
  - It stores our data object in a central location accessible to all components, thus ensuring that there is only “one version of the truth.” This is known as the **state**.
  - Define the specific ways in which components can change that data. These are known as **mutations**.



# VUEx and the index.js file

---

We will be defining code within the index.js file:

```
import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

export default new Vuex.Store({
  {
    state: {},
    mutations: {},
    actions: {},
    modules: {}
  }
});
```

We will concern ourselves primarily with the state and mutations section.

# State: retrieving data

— — —

Here we see how a property defined in the state is retrieved.

```
state: {  
  name: 'Cigar Parties for Dummies',  
  description: 'Host and plan the perfect cigar party for all of your squirrely friends.'  
}
```

index.js

```
<template>  
  <div id="app" class="main">  
    <h1>Product Reviews for {{$store.state.name}}</h1>  
    <p class="description"> {{$store.state.description}}</p>  
    ...
```

VUE Component

# State: Changing Data

— — —

The state's data cannot be changed directly by a component, a mutator must be used, on the bottom left, we have 3 mutators defined:

```
mutations: {                                index.js
  ADD_REVIEW(state, review) {
    state.reviews.unshift(review);
  },
  UPDATE_FILTER(state, filter) {
    state.filter = filter;
  },
  FLIP_FAVORITED(state, reviewToChange) {
    reviewToChange.favorited = !
    reviewToChange.favorited;
  }
},
```

```
methods: {                                  VUE Component
  addNewReview() {
    this.$store.commit("ADD_REVIEW", this.newReview);
    this.resetForm();
  },
```

The method above is calling the mutator on the left (while providing a “payload”).

In this example, within the state, there is an array called reviews, which will be updated with the value of this.newReview.

**Let's see VUEX in action**