Module 3

VUE COMPONENT COMMUNICATION

JAVA– MODULE 3, DAY 13

ELEVATE YOURSELF

Review

RECAP

V-ON

Event to respond to

```
1 <input type="button" value="Cancel" v-on:click="hideQuestionForm">
```

JS Code or Method to Invoke

https://vuejs.org/v2/guide/events.html

V-ON EVENTS

- v-on:**click**=”someMethod”
- v-on:**change**=”someMethod”
- v-on:**submit**=”someMethod”
- v-on:**keyup**=”someMethod”
- v-on:**blur**=”someMethod”
- …
- Basically anything we had before, just in Vue.

V-ON MODIFIERS

- **v-on:click.stop** - Identical to **event.stopPropagation()**
- **v-on:click.prevent** - Identical to **event.preventDefault()**

These also exist, but you'll likely never use them:

- **v-on:click.self** - Ignores bubbled up events from children
- **v-on:click.capture** - Uses capturing instead of bubbling
- **v-on:click.once** - Only care about the first occurrence

Review

METHODS

```
<script>
export default {
  // Methods contain functions which can be invoked from event handlers or other code
  methods: {
    /**
     * Shows the add new question region
     */
    showQuestionForm() {
      this.showAddQuestion = true;
    },
    /**
     * Hides the add new question region
     */
    hideQuestionForm() {
      this.showAddQuestion = false;
    }
  },
  data() {
    return {
      showAddQuestion: false,
    };
  }
};
</script>
```

TECH ELEVATOR

Review

EVENTS & METHODS

```
1 <button v-on:click="handleClick">Delete Product</button>
2
```

```
1 methods: {
2       handleClick() {
3          console.log("Deleting Product...");
4       }
5 },
6
```

V-ON AND METHODS

```
<input type="checkbox"
       id="selectAll"
       v-on:change="selectAllChanged($event)"
       v-bind:checked="areAllSelected" />
```

```
methods: {
  selectAllChanged(e) {
    if (e.target.checked) {
      // Do something if checked
    } else {
      // Do something if unchecked
    }
  },
},
// other code omitted...
```

Review

V-ON AND METHODS

Event argument – $event

```
<input type="checkbox"
       id="selectAll"
       v-on:change="selectAllChanged($event)"
       v-bind:checked="areAllSelected" />
```

```
methods: {
  selectAllChanged(e) {
    if (e.target.checked) {
      // Do something if checked
    } else {
      // Do something if unchecked
    }
  },
},
// other code omitted...
```

In Vanilla JavaScript, you have access to the original DOM event. From there, you can determine where the event originated from. In Vue, the original DOM event is passed to the method by telling the listener to pass it using $event.

Review

CONDITIONAL DISPLAY

Review

V-IF / V-ELSE

```
1 <section id="questionList">
2
3   <BusySpinner v-if="isBusy" />
4
5   <div class="questionContainer" v-else>
6     <QuestionCard v-for="q in filteredQuestions" v-bind:key="q.question" v-bind:question="q"></QuestionCard>
7   </div>
8
9 </section>
```

Review

V-SHOW

```
1 <span class="showAnswer"
2       v-show="!question.isAnswerVisible">
3   {{showAnswerText}}
4 </span>
```

AGENDA

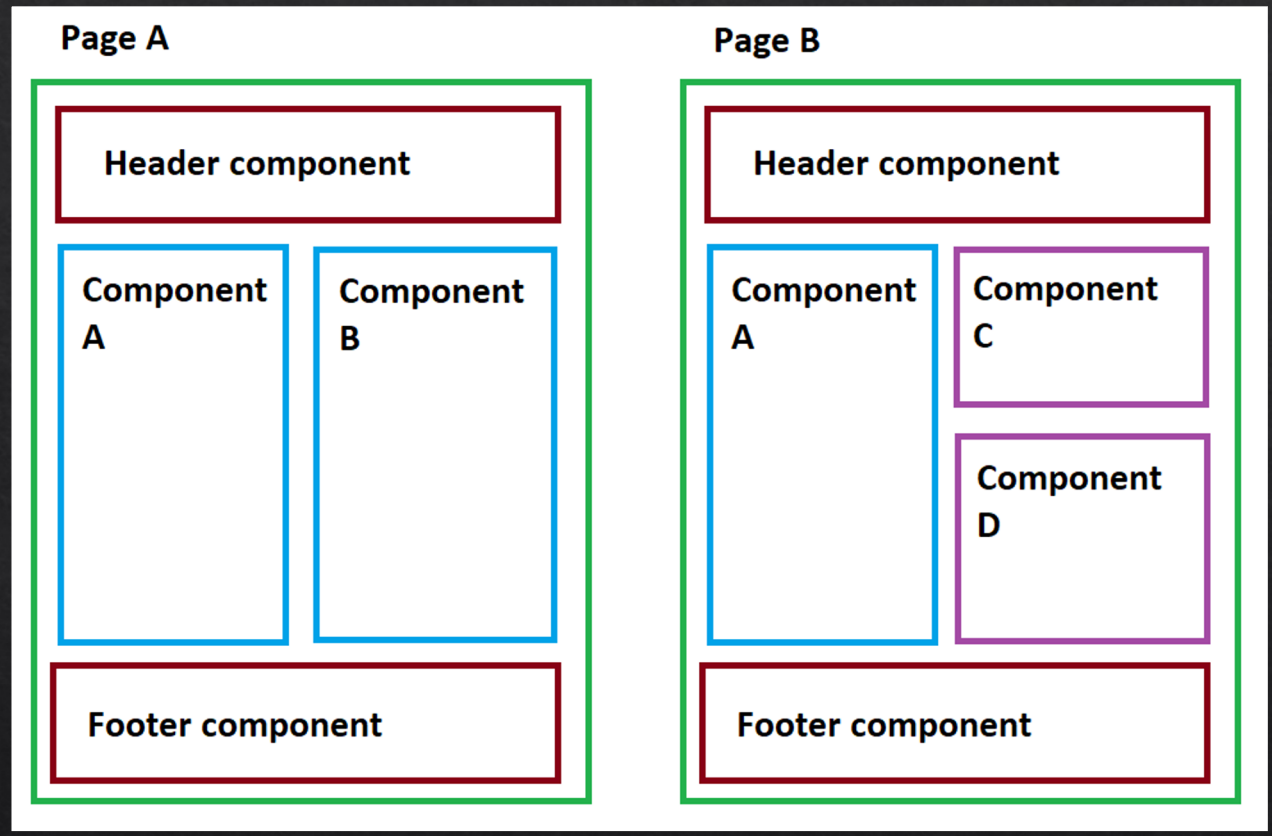
- Sharing State with Vuex
- Single Responsibility Principle
- Child Components & Props

Lecture

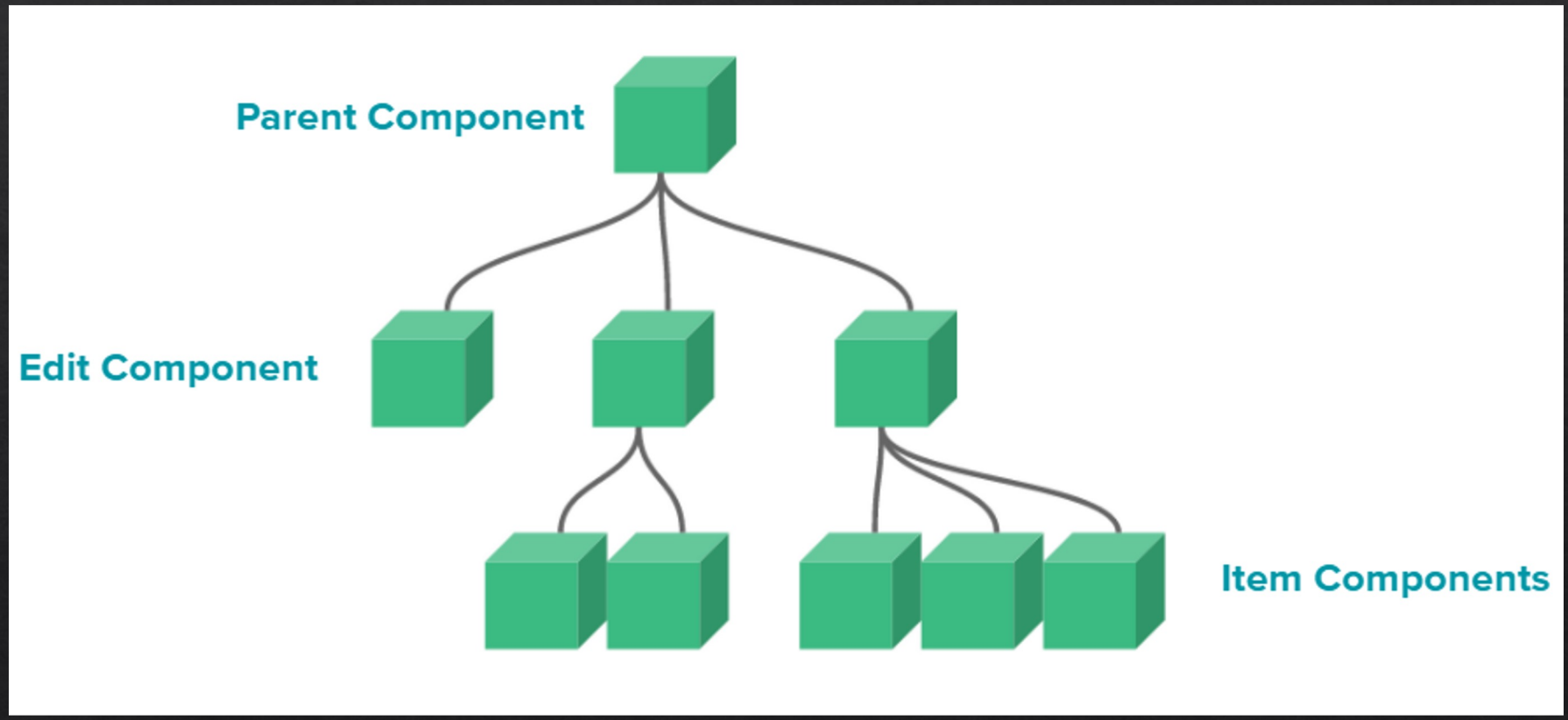
TECH ELEVATOR

Lecture

VUE COMPONENTS



NESTED COMPONENTS



Lecture

SOLID Principles

The **S**ingle Responsibility Principle

The Single Responsibility Principle states that a class should do one thing and therefore it should have only a single reason to change.

The **O**pen-Closed Principle

The Open-Closed Principle requires that **classes should be open for extension and closed to modification.**

Modification means changing the code of an existing class, and extension means adding new functionality.

The **L**iskov Substitution Principle

The Liskov Substitution Principle states that subclasses should be substitutable for their base classes.

The **I**nterface Segregation Principle

Segregation means keeping things separated, and the Interface Segregation Principle is about separating the interfaces.

The **D**ependency Inversion Principle

The Dependency Inversion principle states that our classes should depend upon interfaces or abstract classes instead of concrete classes and functions.

https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/
https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/

TECH ELEVATOR

SINGLE RESPONSIBILITY PRINCIPLE (SRP)

Lecture

Displaying star average

Displaying star summary rating

Displaying review list

Display product information

Adding reviews

Displaying review information

ELEVATE YOURSELF

TECH ELEVATOR

Lecture

COMPONENTS

Lecture

IMPORTING COMPONENTS

```
<!-- Script is the core data and logic associated with the component. It is required -->
<script>
import QuestionList from './QuestionList.vue';
import QuestionSearch from './QuestionSearch.vue';
import AddQuestionForm from './AddQuestionForm.vue';

export default {
  // Contains a list of components which must be included in the template
  components: {
    QuestionList,
    QuestionSearch,
    AddQuestionForm
  },
  // Other code omitted...
};
</script>
```

TECH ELEVATOR

USING COMPONENTS

Lecture

```
1 <template>
2   <main>
3     <section id="questionList">
4       <question-search /> <!-- or QuestionSearch -->
5       <question-list /> <!-- or QuestionList -->
6     </section>
7
8     <section id="closing">
9       <h2>Add Question?</h2>
10      <p v-if="!isAddQuestionVisible">
11        Think we're missing something?
12        <a v-on:click="showAddQuestionForm()">Submit a Question</a> and help us out!
13      </p>
14      <add-question-form v-else /> <!-- or AddQuestionForm -->
15    </section>
16  </main>
17 </template>
```

IMPORTING COMPONENTS

To import and use a component within another component, perform these three steps:

1.Import the component.

2.Declare the component in the components object.

3.Use the component in your markup (template).

```
<!--App.vue-->
<template>
  <the-header></the-header>
  <!-- Step 3 -->
</template>

<script>
  import TheHeader from './components/TheHeader'; // Step 1

  export default {
    components: {
      TheHeader // Step 2
    }
  };
</script>
```

TECH ELEVATOR

Lecture

PROPERTIES

PROPS

```
export default {
  name: 'blog-posts',
  props: ['posts']
};
```

```
<template>
  <article v-for="post in posts" v-bind:key="post.id"></article>
</template>

<script>
  export default {
    name: 'blog-posts',
    props: ['posts']
  };
</script>
```

PASSING PROPS

```
1 <template>
2   <div id="app">
3     <!-- These are custom components we've created -->
4     <AppHeader />
5     <questions />
6     <app-footer myPropName="Any problem can be solved with another layer of abstraction" />
7   </div>
8 </template>
```

TECH ELEVATOR

BINDING PROPS

Lecture

```
<template>
    <div>
        <question-card v-for="item of filteredItems"
                       v-bind:key="item.id"
                       v-bind:cardData="item" /> <!-- Binds a prop named 'cardData' to item
    </div>
</template>
```

TECH ELEVATOR

Lecture

GLOBAL COMPONENTS

Lecture

DATASTORE

What about when multiple components need to both access and modify data across your application?

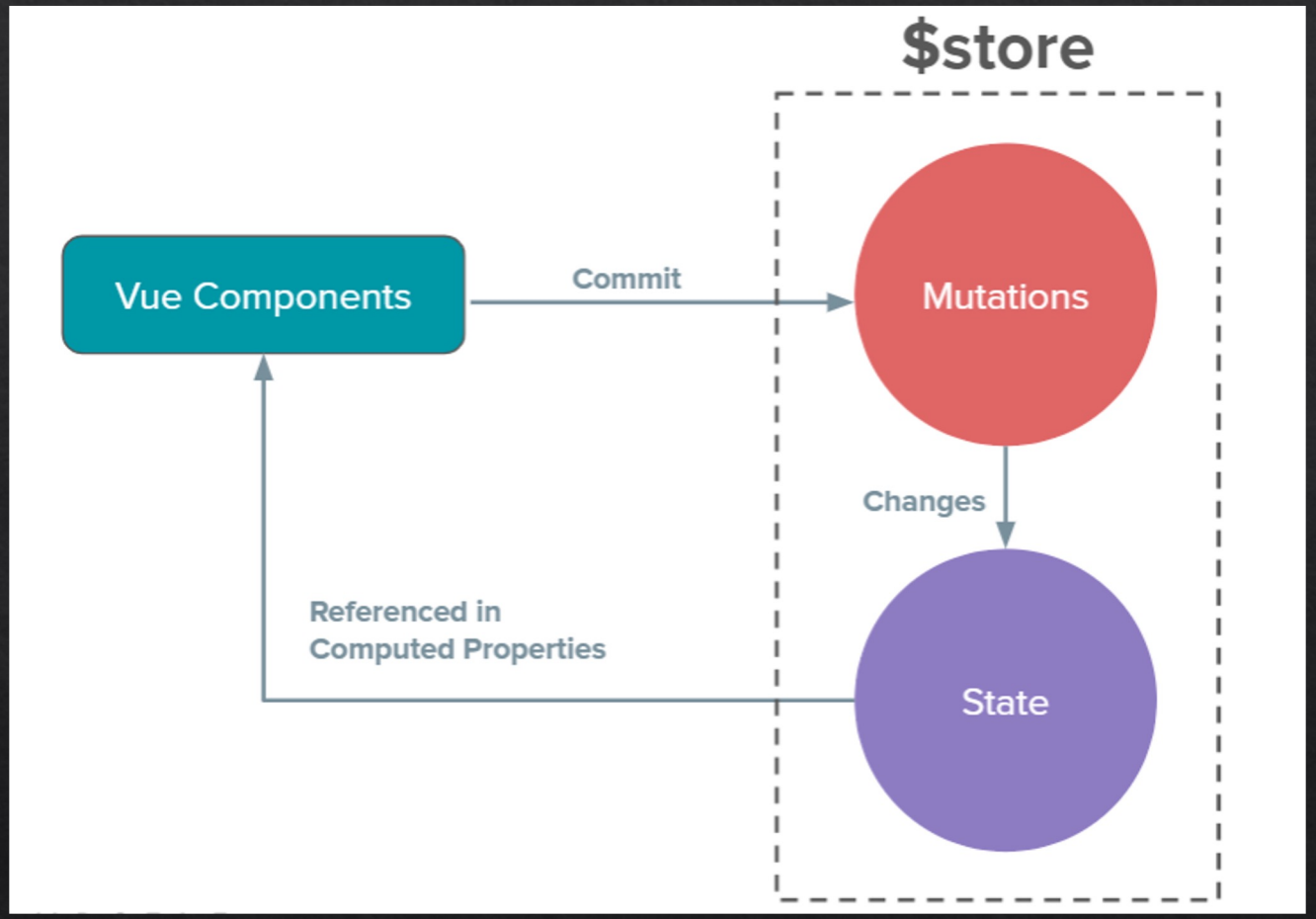
When multiple components need to both access and modify data across your application, it's best for all of your application data to live in one central location that your components can connect to. This concept is common in programming and is typically called a **datastore**.

TECH ELEVATOR

Lecture

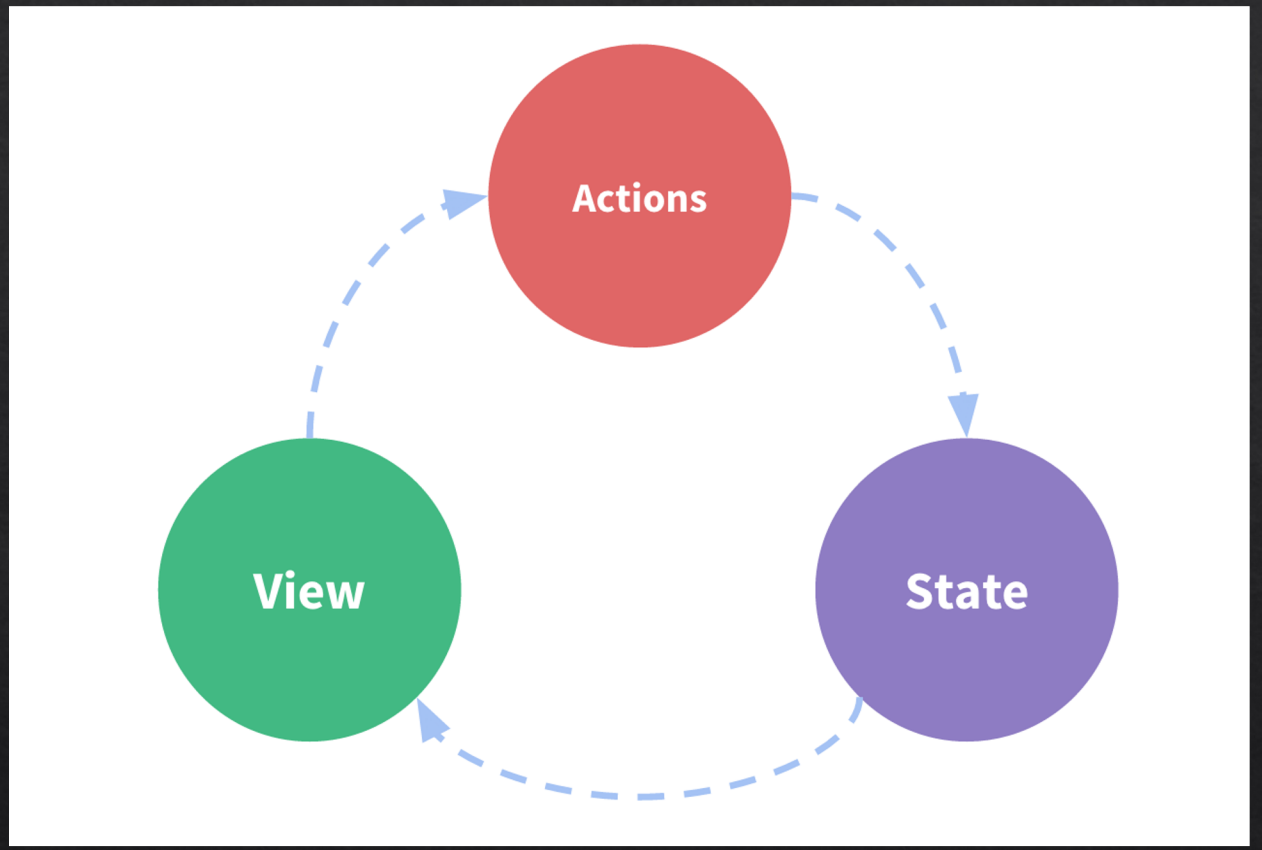
VUEX

VUEX


$store
Vue Components
Commit
Mutations
Changes
Referenced in
Computed Properties
State

Lecture

VUEX STATE



TECH ELEVATOR

VUEX MUTATIONS

Lecture

```
export default new Vuex.Store({
  state: {
    posts: [
      {
        id: 1,
        title: 'My First Post',
        content: '<p>This is my first post</p>'
      },
      {
        id: 2,
        title: 'My Second Post',
        content: '<p>This is my second post</p>'
      }
    ]
  },
  mutations: {}, // MUTATIONS GO HERE
  actions: {},
  modules: {}
});
```

STORE DEFINITION

```
export default new Vuex.Store({
  // State contains the global application state. Think of it as app-wide data
  state: {
    isAddQuestionVisible: false,
  },

  // Mutations are used to make discrete changes to state from a central place
  mutations: {
    SHOW_ADD_QUESTION(state) {
      state.isAddQuestionVisible = true;
    },
    HIDE_ADD_QUESTION(state) {
      state.isAddQuestionVisible = false;
    },
  },
})
```

WORKING WITH THE STORE

```
export default {
  // Methods contain functions which can be invoked from event handlers or other code
  methods: {
    /**
     * Shows the add question form.
     */
    showAddQuestionForm() {
      this.$store.commit('SHOW_ADD_QUESTION');
    },
  },
  // Computed contains the Vue.js equivalent of getters that rely on other data to compute results.
  // Think of computed as computed or auto-calculated properties that are bound to from the template.
  computed: {
    isAddQuestionVisible() {
      return this.$store.state.isAddQuestionVisible;
    }
  }
};

```

SENDING PARAMETERS

Lecture

```
export default {
  // Methods contain functions which can be invoked from event handlers or other code
  methods: {
    someMethod() {
      // Call the SOME_MUTATOR_NAME mutator with a payload of 42
      this.$store.commit('SOME_MUTATOR_NAME', 42);
    },
  },
  // Other members omitted...
};
```

RECEIVING PARAMETERS

```
1 export default new Vuex.Store({
2   // State contains the global application state. Think of it as app-wide data
3   state: {
4     isAddQuestionVisible: false,
5     someVariable = 0,
6   },
7
8   // Mutations are used to make discrete changes to state from a central place
9   mutations: {
10    SHOW_ADD_QUESTION(state) {
11      state.isAddQuestionVisible = true;
12    },
13    HIDE_ADD_QUESTION(state) {
14      state.isAddQuestionVisible = false;
15    },
16    SOME_MUTATOR_NAME(state, payload) {
17      state.someVariable = payload;
18    },
19  },
20 })
21
```

Lecture

ADDITIONAL RESOURCES

- Bootcamp OS
- Vue.js Documentation - (https://vuejs.org/v2/guide/)
- Vue CLI – (https://cli.vuejs.org/)

TECH ELEVATOR

Module 3

Vue Component Communication

New Tools! Let's Code!

Vue.js
Component_Communication

Developer's Toolbox

Exercise HW