Module 3 Day 13:

Vue Component Communication

Can you?

- Identify candidate components in a web page/application
- Explain the "Single Responsibility Principle"
- Implement a simple SPA that utilizes components that work together
- Implement a "child" component that's passed data from the parent using props
- Create child components from an array in a parent component
- Access data stored in Vuex from multiple components
- Modify data stored in Vuex using "mutations"

Single Responsibility Principle

- The Single Responsibility Principle (SRP) states that each component in the application should only handle one job.
- If a component isn't trying to handle an entire dashboard but is instead focused on just one graph on that dashboard, that component is easier to test, easier to maintain, and easier to reuse in another context..

Naming Components

- According to the Vue style guide, "component names should always be multi-word, except for root App components, and built-in components provided by Vue, such as <transition> or <component>."
- Single name Components like Header should be named TheHeader to follow this convention.

Passing Data to Child Components

- Components can use other Components.
- A Component can expect a property to be passed to it from another component by specifying the expected property in a `props` property.
    - `props: ['reviewData']`
- Properties passed via `props` can be used like any properties defined in the Component itself.
- Parent Components can pass data to a Component with a `props` property by binding an attribute with the name of the expected property:
    - `<review-display v-bind:review-data="review"></review-display>`

Prop Names

- Multi-word **`props`** should be defined using camelCase
  - **`props: ['reviewData']`**
- When passing an attribute via **`v-bind`**, multi-word props should be specified in kebab case:
  - **`<review-display v-bind:review-data="review"></review-display>`**

Component communication using Vuex

- Vuex is a **state management pattern** and library for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the **state** (data) can only be **mutated** (changed) in a predictable fashion.
- state and state management refer to data within an application and how it is managed.
- Vuex data store is contained in a /`src/store/index.js` file.
- If more than one Component needs access to the same data, this is where it would be stored.

Component communication using Vuex

- Shared data, available for ***any*** Component to access, is contained in the data stores' state property.

```
export default new Vuex.Store({
  state: {
    name: 'Cigar Parties for Dummies',
    description: 'Host and plan the perfect cigar party for all of your squirrelly friends.',
    filter: 0,
    reviews: [
      {
        reviewer: 'Malcolm Gladwell',
        (property) favorited: boolean
        "favorited": Unknown word. cSpell
        Peek Problem (Alt+F8)    Quick Fix... (Ctrl+.)
        I can see that. Pages kept together with glue and there's writing on it, in some language.
        favorited: false
      },
      {
        reviewer: 'Tim Ferriss',
        title: 'Had a cigar party started in less than 4 hours.',
        review:
          "It should have been called the four hour cigar party. That's amazing. I have a new idea for muse because of this.",
        rating: 4,
        favorited: false
      }
    ],
  },
  mutations: {
  },
  actions: {
  },
  modules: {
  },
  strict: true
})
```

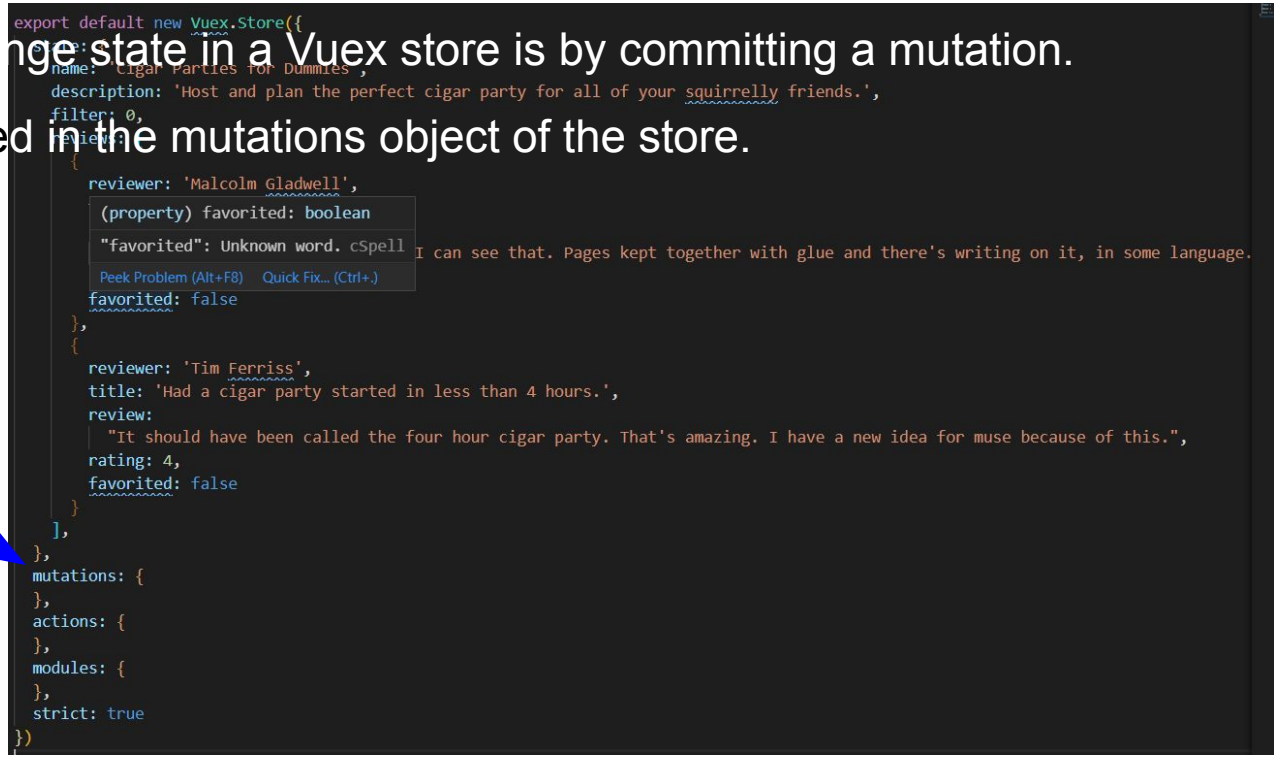
Accessing Data from the Store

- Data in the store can be accessed via the `$store` object in a Component.
- Since the data portion of the store is in the state property, properties can be accessed via `$store.state`
  - `$store.state.reviews`
- Accessing the store in your component's methods and computed properties can be done like any other property with the ***this*** keyword:
  - `this.$store.state.reviews`

Vuex Mutations

- The only way to change state in a Vuex store is by committing a mutation.
- Mutations are defined in the mutations object of the store.

Mutations will be defined here.



Defining Mutations

- To define a mutation that adds a new post to your list of posts, you'd start by creating a function called ADD_POST(). The function is where you perform state modifications, and it receives the state as the first argument.

```
mutations: {
  ADD_REVIEW(state, review) {
    state.reviews.unshift(review);
  }
},
```

Using Mutations

- Mutation handlers can't be called directly.
- To call a mutation we use **`$store.commit`** with the mutation name
  - **`$store.commit('ADD_REVIEW')`**
- An additional object (known as the payload) can also be passed
  - **`$store.commit('ADD_REVIEW', review)`**
- When Vuex store is created with the **`strict`** property set to **`true`**, an error is thrown if code attempts to modify store state data directly rather than through mutations.