Vue Component Communication

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 property in the **`props`** property.
    - **`prop: ['posts']`**
- Properties passed via **`props`** can be used like any properties defined in the Component itself.
- Other Components can pass data to a Component with a **`props`** property by binding an attribute with the name of the expected property:
    - **`<blog-posts v-bind:posts="posts"></blog-posts>`**

Prop Names

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

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 it would be stored.

Component communication using Vuex

- Shared data that any Component can access are contained in the data stores's state property.

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

Vue.use(Vuex);

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: {},
  actions: {},
  modules: {}
});
```

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`**
    - **`$store.state.posts`**
- As with properties, to access the store in your methods and computed properties you have to use this:
    - **`this.$store.state.posts`**

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.

```
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: {}
});
```

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_POST(state) {
    state.posts.push({
      id: 3,
      title: 'My Third Post',
      content: '<p>This is my third post</p>'
    });
  }
},
```

Using Mutations

- Mutation handlers can't be called directly.
- To call a mutation we use **`$store.commit`** with the mutation type
  - **`$store.commit('ADD_POSTS')`**
- An addition object (known as the payload) can be passed
  - **`$store.commit('ADD_POSTS', posts)`**
- 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.