CHAPTER 6

State Management with Vuex

Until this point in the book, all data has been stored in our components. We hit an API, and we store the returned data on the data object. We bind a form to an object, and we store that object on the data object. All communication between components has been done using events (to go from child to parent) and props (to go from parent to child). This is good for simple cases, but in more complicated applications, it won't suffice.

Let's take a social network app—specifically, messages. You want an icon in the top navigation to display the number of messages you have, and then you want a messages pop-up at the bottom of the page that will also tell you the number of messages you have. Both components are nowhere near each other on the page, so linking them using events and props would be a nightmare: components that are completely unrelated to notifications will have to be aware of the events to pass them through. The alternative is, instead of linking them together to share data, you could make separate API requests from each component. That would be even worse! Each component would update at different times, meaning they could be displaying different things, and the page would be making more API requests than it needed to.

vuex is a library that helps developers manage their application's state in Vue applications. It provides one centralized store that you can use throughout your app to store and work with global state, and gives you the ability to validate data going in to ensure that the data coming out again is predictable and correct.

Installation

You can use vuex via a CDN. Just add the following:

```
<script src="https://unpkg.com/vuex"></script>
```

Alternatively, if you're using npm, you can install vuex by using `npm install --save vuex`. If you're using a bundler such as webpack, then just as with vue-router, you have to call `Vue.use()`:

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

Vue.use(Vuex);
```

Then you need to set up your store. Let's create the following file and save it as *store/index.js*:

```
import Vuex from 'vuex';

export default new Vuex.Store({
  state: {}
});
```

For now, that's just an empty store: we'll add to it throughout the chapter.

Then, import it in your main app file and add it as a property when creating the Vue instance:

```
import Vue from 'vue';
import store from './store';

new Vue({
  el: '#app',
  store,
  components: {
    App
  }
});
```

You've now added the store to your app and you can access it using `this.$store`. Let's look at the concepts of vuex and then we'll look at what you can do with `this.$store`.

Concept

As mentioned in the introduction to this chapter, vuex can be required when complex applications require more than one component to share state.

Let's take a simple component written without vuex that displays the number of messages a user has on the page:

```
const NotificationCount = {
  template: `<p>Messages: {{ messageCount }}</p>`,
  data: () => ({
    messageCount: 'loading'
  }),
  mounted() {
    const ws = new WebSocket('/api/messages');

    ws.addEventListener('message', (e) => {
      const data = JSON.parse(e.data);
      this.messageCount = data.messages.length;
    });
  }
};
```

It's pretty simple. It opens a websocket to */api/messages*, and then when the server sends data to the client—in this case, when the socket is opened (initial message count) and when the count is updated (on new messages)—the messages sent over the socket are counted and displayed on the page.

In practice, this code would be much more complicated: there's no authentication on the websocket in this example, and it is always assumed that the response over the websocket is valid JSON with a `messages` property that is an array, when realistically it probably wouldn't be. For this example, this simplistic code will do the job.

We run into problems when we want to use more than one of the `NotificationCount` components on the same page. As each component opens a websocket, it opens unnecessary duplicate connections, and because of network latency, the components might update at slightly different times. To fix this, we can move the websocket logic into vuex.

Let's dive right in with an example. Our component will become this:

```
const NotificationCount = {
  template: `<p>Messages: {{ messageCount }}</p>`,
  computed: {
    messageCount() {
      return this.$store.state.messages.length;
    }
  }
  mounted() {
    this.$store.dispatch('getMessages');
  }
};
```

And the following will become our vuex store:

```
let ws;

export default new Vuex.Store({
  state: {
    messages: [],
  },
  mutations: {
    setMessages(state, messages) {
      state.messages = messages;
    }
  },
  actions: {
    getMessages({ commit }) {
      if (ws) {
        return;
      }

      ws = new WebSocket('/api/messages');

      ws.addEventListener('message', (e) => {
        const data = JSON.parse(e.data);
        commit('setMessages', data.messages);
      });
    }
  }
});
```

Now, every notification count component that is mounted will trigger `getMessages`, but the action checks whether the websocket exists and opens a connection only if there isn't one already open. Then it listens to the socket, committing changes to the state, which will then be updated in the notification count component as the store is reactive—just like most other things in Vue. When the socket sends down something new, the global store will be updated, and every component on the page will be updated at the same time.

Throughout the rest of the chapter, I'll introduce the individual concepts you saw in that example—state, mutations, and actions—and explain a way we can structure our vuex modules in large applications to avoid having one large, messy file.

State and State Helpers

First, let's look at state. *State* indicates how data is stored in our vuex store. It's like one big object that we can access from anywhere in our application—it's the *single source of truth*.

Let's take a simple store that contains only a number:

```
import Vuex from 'vuex';

export default new Vuex.Store({
  state: {
    messageCount: 10
  }
});
```