Routing guards and persisted data

In this section we will

- Stop anyone being able to access chat directly via URL if they haven't specified a username
- Send an auth command with the username specified at login

Understanding Vuex

Vuex is a state management pattern + library for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion – vuex.vuejs.org

Consider Vuex an in memory database for your application. The upside, is that data stored in Vuex is reactive. Meaning if you have 2 components that both rely on the same stored data, if you update the value of that data in one component the other component's data is automatically updated to be the same.

You will lose all information stored in Vuex when you navigate away from your application. There are extensions you can install that will persist this information (In local storage, for example). From this, you can see how powerful Vuex can be.

Traditionally, Vuex is a good fit for larger scale applications. It is a bit overkill for what we are doing with our chat client. However, it's good to get an understanding of staple tooling within Vue.

We are going to store the username specified in the login component and retrieve it in the chat component.

1. In ***src/store/user-store.ts***
    - Notice there are stubbed entries for ***getters*** and ***mutations***
    - Inside ***getters***, create an empty method called ***getUsername***. This will take a parameter named ***state*** of type ***IUserState***
    - Inside ***getUsername*** return ***state.username***
    - Inside ***mutations***, create an empty method called ***setUsername***. This will take a parameter named ***state*** of type ***IUserState*** and a parameter ***value*** of type ***string***.
    - Inside ***setUsername*** set ***state.username*** equal to ***value***.

2. In ***src/views/login/login.component.ts***
    - Create a ***private property*** called ***username*** of type ***string.*** Initialise it to an ***empty string***
    - Inside ***buttonClicked*** insert the following code before the router navigation
        i. this.$store.commit("username/setUsername", this.username);

3. in ***src/views/login/login.html***
    - Bind your component's ***username*** property to your ***input*** field using ***v-model***

4. In ***src/views/chat/chat.component.html***
    - Replace the initialisation of your ***AuthCommand*** with the following code
        i. new AuthCommand(this.$store.getters["username/getUsername"])

5. Run your application. When you type a username on the login screen, you should now see the auth command issued with that username.

Setting an auth guard

6. Inside ***src/router/index.ts***
    - Fill out the ***beforeEnter*** function for the chat route. Do this by retrieving the ***username*** from the store. If the ***username*** is null, undefined, or its length is 0 call ***next(from),*** else call ***next()***

    With this in place, we've made sure that when we hit our chat component, we always have a username available to connect to. In a real-world scenario, we would likely have some kind of validation in place to show the user as well. These are the building blocks you can start to build on top of to create an enterprise level application.

    In your browser, log in to your application and enter the chat screen. When you refresh the web page, you will see you are now redirected back to login. A refresh of the web page destroys all state in our Vuex store because it is not persisted.

Congratulations. You've finished part 2!