Final Vue Capstone Project Seed

This is the Vue starter project for the final capstone. This document walks you through how to set up and run the project. It also explains the project's features, such as Vue Router, Vuex, and authentication.

Project setup

The first thing you'll need to do is to download any dependencies by running this command:

```
npm install
```

Next take a moment to review the `.env` file that's located in the root of the project. You can store environment variables that you want to use throughout your application in this file. When you open it, it'll look like this:

```
# Java
VUE_APP_REMOTE_API=http://localhost:9000
```

Note: The Java Spring Boot application is configured to run on port 9000 instead of 8080.

Start your Vue application with the following command:

```
npm run serve
```

Authentication

When you first run the project and visit the base URL, you're taken to a login page. This is because the home route `/` is secured by default. If you look in `/src/router/index.js`, you'll see the following code:

```
router.beforeEach((to, from, next) => {
  // Determine if the route requires Authentication
  const requiresAuth = to.matched.some(x => x.meta.requiresAuth);

  // If it does and they are not logged in, send the user to "/login"
  if (requiresAuth && store.state.token === '') {
    next("/login");
  } else {
    // Else let them go to their next destination
    next();
  }
});
```

This is a feature of Vue Router called Navigation Guards. You may not have learned about this in class, so take some time to read through the documentation to learn what they are and how they work.

The above code runs before each route. It first checks to see if the route requires authentication that is defined per route using the meta object key `requiresAuth`.

In the following configuration, you must be authenticated to view the home route while anyone can visit the login, logout, and registration routes:

```
const router = new Router({
  mode: 'history',
  base: process.env.BASE_URL,
  routes: [
    {
      path: '/',
      name: 'home',
      component: Home,
      meta: {
        requiresAuth: true
      }
    },
    {
      path: "/login",
      name: "login",
      component: Login,
      meta: {
        requiresAuth: false
      }
    },
    {
      path: "/logout",
      name: "logout",
      component: Logout,
      meta: {
        requiresAuth: false
      }
    },
    {
      path: "/register",
      name: "register",
      component: Register,
      meta: {
        requiresAuth: false
      }
    },
  ]
})
```

Next, the navigation guard checks to see if the route requires authentication and if an authentication token exists.

If authentication is not required, *or* the authentication token does exist—meaning it isn't an empty string—the user is routed to the requested route.

However, if authentication is required *and* the authentication token doesn't exist—meaning it's an empty string—the user is redirected to the `/login` route:

```
// If it does and they are not logged in, send the user to "/login"
if (requiresAuth && store.state.token === '') {
  next("/login");
} else {
  // Else let them go to their next destination
  next();
}
```

> Note: the application stores the current user (if any) and their authentication token in a centralized store using Vuex.

Vuex

The state for this application is stored in `/store/index.js` using Vuex. The state object has two values: token and user. When you log in, the back-end service returns an authentication token along with your user credentials.

The authentication token is sent in the `Authorization` header to verify your identify. To persist this token when the application is closed or the page is refreshed, you'll store the token in local storage.

The default token either comes from local storage or it is set to an empty string. As you learned in the previous section, if the route requires authentication and this token is empty, it redirects the user to the login page:

```
const currentToken = localStorage.getItem('token')

export default new Vuex.Store({
  state: {
    token: currentToken || '',
    user: currentUser || {}
  },
```

Login

When you reach the `/login` route, you'll see a bare login page. This is intentional. It's up to you to style this page to fit within your application.

When you fill in a username and password and click the "Sign In" button, the method `login()` is called. The `login()` method uses the `/src/services/AuthService.js` to send a `POST` request to your API's `/login` route.

If you look at `AuthService`, you'll notice that there's no base URL set for Axios:

```
import axios from 'axios';

export default {

  login(user) {
    return axios.post('/login', user)
  }

}
```

This is because this value is set in `/src/main.js` and the value comes from the `.env` property file you saw earlier:

```
axios.defaults.baseURL = process.env.VUE_APP_REMOTE_API;
```

If you get a successful response (200), it contains the authentication token and user object. You'll set these in Vuex by committing mutations:

```
login() {
  authService
    .login(this.user)
    .then(response => {
      if (response.status == 200) {
        this.$store.commit("SET_AUTH_TOKEN", response.data.token);
        this.$store.commit("SET_USER", response.data.user);
        this.$router.push("/");
      }
    })
}
```

When you call the `SET_AUTH_TOKEN` mutation, several things happen.

First, you set the `state.token` value to what was returned from the API's `/login` method. Next, you store that same value in local storage so that it persists across refreshes. Finally, you set the `Authorization` header in Axios so that every subsequent request contains the token. This way, you don't have to manually do this on every request:

```
mutations: {
  SET_AUTH_TOKEN(state, token) {
    state.token = token;
    localStorage.setItem('token', token);
    axios.defaults.headers.common['Authorization'] = `Bearer ${token}`
  }
}
```

Once the `login()` method finishes updating the store by committing the mutations, it forwards the user back to the homepage. They'll be able to see the homepage because they're authenticated.

Logout

There's a logout link in `App.vue` that forwards the user to the `/logout` route. When the user reaches this route, you'll commit this mutation in the store called `LOGOUT`:

```
<template>
  <h1>Logout</h1>
</template>

<script>
export default {
  created() {
    this.$store.commit("LOGOUT");
    this.$router.push("/login");
  }
};
</script>
```

When the mutation is called, the token is removed from local storage, the token and user state are cleared, and the user is redirected back to the homepage. The homepage then forwards the user to the login page because they're no longer logged in:

```
mutations: {
  LOGOUT(state) {
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    state.token = '';
    state.user = {};
  }
}
```

Register

When you reach the `/register` route, you'll see a bare registration page. Like the login page, this is intentional. You'll need to style this page to fit within your application.

When you fill in a username, password, confirm the password role, and click the "Create Account" button, the method `register()` is called. This calls the `register()` method in `/src/services/AuthService.js`. This passes your user details to your back-end application's REST API to create a new user:

```
methods: {
register() {
  // ...
  authService
```

```
      .register(this.user)
      .then(response => {
        if (response.status == 201) {
          this.$router.push({
            path: "/login",
            query: { registration: "success" }
          });
        }
      })
    // ...
  }
```