Vue 3 Authentication with JWT, Vuex, Axios and Vue Router

Last modified: September 6, 2021 (https://www.bezkoder.com/vue-3-authentication-jwt/) bezkoder (https://www.bezkoder.com/author/bezkoder/) Security (https://www.bezkoder.com/category/security/), Vue.js (https://www.bezkoder.com/category/vue/)

In this tutorial, we're gonna build Vue 3 Authentication & Authorization example with JWT, Vuex, Axios, Vue Router and VeeValidate. I will show you:

- JWT Authentication Flow for User Signup & User Login
- Project Structure for Vue 3 Authentication with Vuex 4 & Vue Router 4
- How to define Vuex Authentication module
- Creating Vue 3 Authentication Components with Vuex Store
- Reactive Form Validation with VeeValidate 4
- Vue 3 Components for accessing protected Resources
- How to add a dynamic Navigation Bar to Vue 3 App

Let's explore together.

Related Post:
– In-depth Introduction to JWT-JSON Web Token (https://bezkoder.com/jwt-json-web-token/)
– Vue 3 CRUD example with Axios & Vue Router (https://bezkoder.com/vue-3-crud/)

Fullstack:
– Spring Boot + Vue.js: Authentication with JWT & Spring Security Example (https://bezkoder.com/spring-boot-vue-js-authentication-jwt-spring-security/)
– Node.js Express + Vue.js: JWT Authentication & Authorization example (https://bezkoder.com/node-express-vue-jwt-auth/)

Contents [hide]

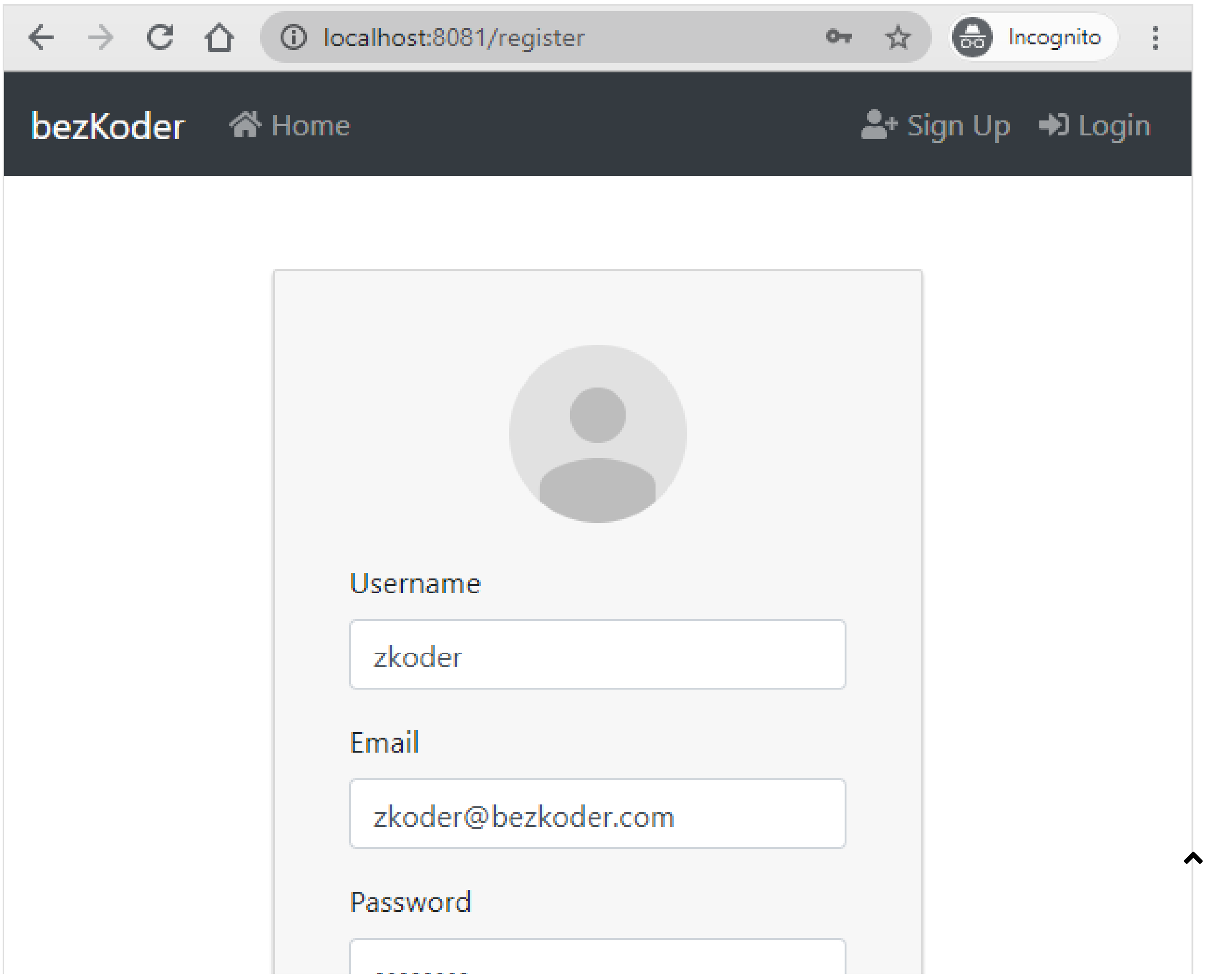
Overview of Vue 3 Authentication with JWT example

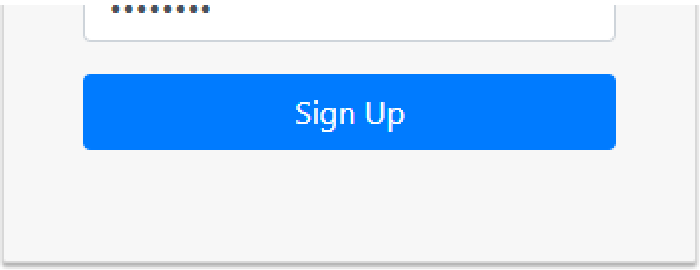
We will build a Vue 3 application in that:

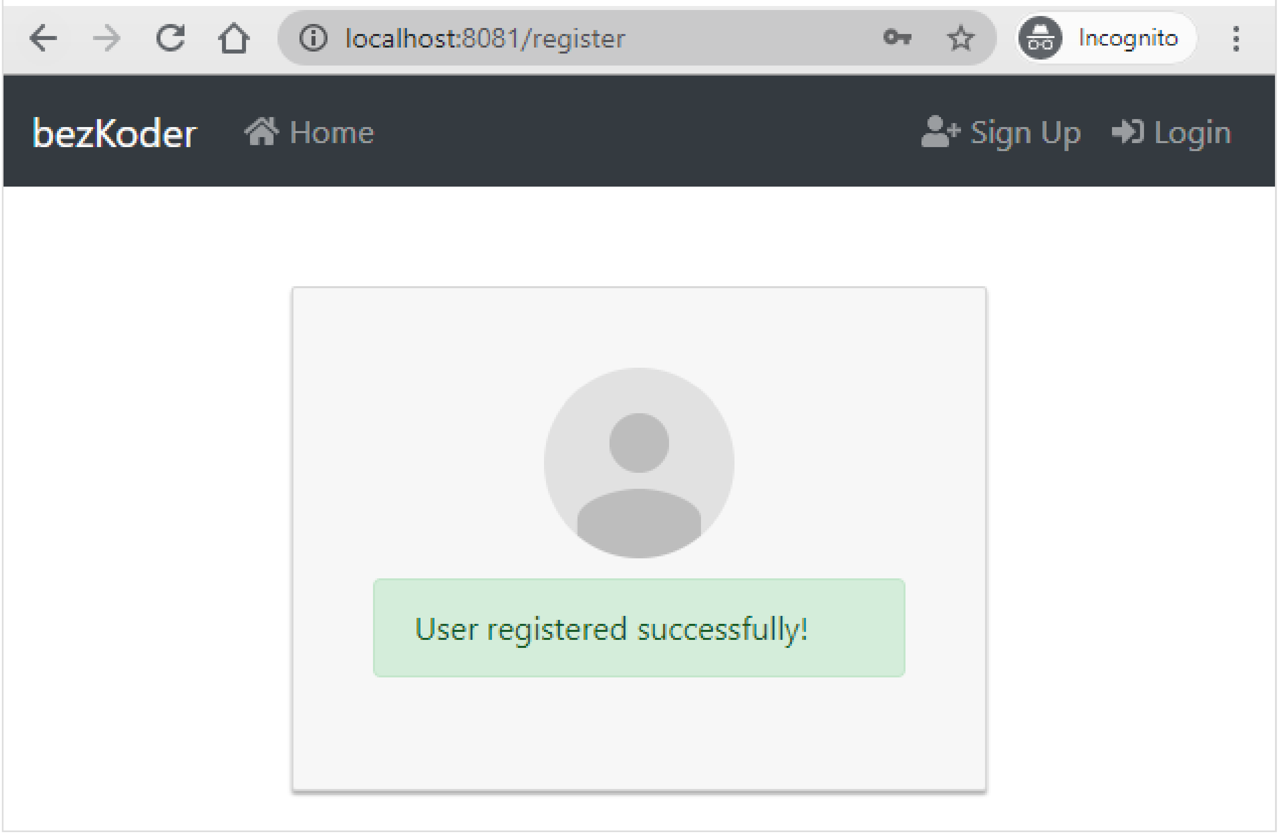
- There are Login/Logout, Signup pages.
- Form data will be validated by front-end before being sent to back-end.
- Depending on User's roles (admin, moderator, user), Navigation Bar changes its items automatically.

Screenshots

– Signup Page:







User registered successfully!

– Form Validation could look like this:

– Login Page & Profile Page (for successful Login):

– Navigation Bar for Admin account:

You will need to add Refresh Token, more details at:
Vue 3 Refresh Token with Axios and JWT example (https://www.bezkoder.com/vue-3-refresh-token/)

Demo

This is full Vue JWT Authentication App demo (with form validation, check signup username/email duplicates, test authorization with 3 roles: Admin, Moderator, User). In the video, we use Spring Boot for back-end REST APIs.

In the video, we use Vue 2 and VeeValidate 2, but logic and UI are the same as this tutorial.

Flow for User Registration and User Login

For JWT Authentication, we're gonna call 2 endpoints:

- POST `api/auth/signup` for User Registration
- POST `api/auth/signin` for User Login

You can take a look at following flow to have an overview of Requests and Responses Vue Client will make or receive.

Vue Client must add a JWT to HTTP Authorization Header before sending request to protected resources.

You can find step by step to implement these back-end servers in following tutorial:

- Spring Boot JWT with Spring Security & MySQL (https://bezkoder.com/spring-boot-jwt-authentication/)
- Spring Boot JWT with Spring Security & PostgreSQL (https://bezkoder.com/spring-boot-security-postgresql-jwt-authentication/)
- Spring Boot JWT Authentication with Spring Security, MongoDB (https://bezkoder.com/spring-boot-jwt-auth-mongodb/)
- Node.js JWT Authentication & Authorization with MySQL (https://bezkoder.com/node-js-jwt-authentication-mysql/)
- Node.js JWT Authentication & Authorization with MongoDB (https://bezkoder.com/node-js-mongodb-auth-jwt/)
- Node.js JWT Authentication & Authorization with PostgreSQL (https://bezkoder.com/node-js-jwt-authentication-postgresql/)

Vue App Component Diagram with Vuex & Vue Router

Now look at the diagram below.

Let’s think about it.

– The `App` component is a container with `Router`. It gets app state from Vuex `store/auth`. Then the navbar now can display based on the state. `App` component also passes state to its child components.

– `Login` & `Register` components have form for submission data (with support of `vee-validate`). We call Vuex store `dispatch()` function to make login/register actions.

– Our Vuex actions call `auth.service` methods which use `axios` to make HTTP requests. We also store or get **JWT** from Browser Local Storage inside these methods.

– `Home` component is public for all visitor.

– `Profile` component get `user` data from its parent component and display user information.

– `BoardUser`, `BoardModerator`, `BoardAdmin` components will be displayed by Vuex state `user.roles`. In these components, we use `user.service` to get protected resources from API.

– `user.service` uses `auth-header()` helper function to add JWT to HTTP Authorization header.
`auth-header()` returns an object containing the JWT of the currently logged in user from Local Storage.

Technology

We will use these modules:

- vue 3
- vue-router 4
- vuex 4
- axios: 0.21.1
- vee-validate 4
- bootstrap 4
- vue-fontawesome 3

Project Structure

This is folders & files structure for our Vue 3 Authentication & Authorization with JWT application:

With the explanation in diagram above, you can understand the project structure easily.

Setup Vue 3 Project

Open cmd at the folder you want to save Project folder, run command:
`vue create vue-3-authentication-jwt`

You will see some options, choose **Default ([Vue 3] babel, eslint)**.

After the project is ready, run following command to install neccessary modules:

```
npm install vue-router@4
npm install vuex@4
npm install vee-validate@4 yup
npm install axios
npm install bootstrap@4 jquery popper.js
npm install @fortawesome/fontawesome-svg-core @fortawesome/free-solid-svg-icons @fortawes
```

After the installation is done, you can check `dependencies` in *package.json* file.

```
"dependencies": {
  "@fortawesome/fontawesome-svg-core": "^1.2.35",
  "@fortawesome/free-solid-svg-icons": "^5.15.3",
  "@fortawesome/vue-fontawesome": "^3.0.0-3",
  "axios": "^0.21.1",
  "bootstrap": "^4.6.0",
  "core-js": "^3.6.5",
  "jquery": "^3.6.0",
  "popper.js": "^1.16.1",
  "vee-validate": "^4.3.5",
  "vue": "^3.0.0",
  "vue-router": "^4.0.6",
  "vuex": "^4.0.0",
  "yup": "^0.32.9"
},
```

In **src** folder, we create **plugins**/*font-awesome.js* file with following code:

```
import { library } from "@fortawesome/fontawesome-svg-core";
import { FontAwesomeIcon } from "@fortawesome/vue-fontawesome";
import {
  faHome,
  faUser,
  faUserPlus,
  faSignInAlt,
  faSignOutAlt,
} from "@fortawesome/free-solid-svg-icons";

library.add(faHome, faUser, faUserPlus, faSignInAlt, faSignOutAlt);

export { FontAwesomeIcon };
```

Open **src**/*main.js*, modify code inside as below:

```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";
import store from "./store";
import "bootstrap";
import "bootstrap/dist/css/bootstrap.min.css";
import { FontAwesomeIcon } from './plugins/font-awesome'

createApp(App)
  .use(router)
  .use(store)
  .component("font-awesome-icon", FontAwesomeIcon)
  .mount("#app");
```

You can see that we import and apply:
– `store` for Vuex (implemented later in **src**/**store**)
– `router` for Vue Router (implemented later in **src**/*router.js*)
– `bootstrap` with CSS
– `vue-fontawesome` for icons (used later in `nav`)

Create Services

We create two services in *src/services* folder:

services

- auth-header.js
- auth.service.js *(Authentication service)*
- user.service.js *(Data service)*

Authentication service

The service provides three important methods with the help of axios for HTTP requests & reponses:

- `login()` : POST {username, password} & save `JWT` to Local Storage
- `logout()` : remove `JWT` from Local Storage
- `register()` : POST {username, email, password}

```
import axios from 'axios';

const API_URL = 'http://localhost:8080/api/auth/';

class AuthService {
  login(user) {
    return axios
      .post(API_URL + 'signin', {
        username: user.username,
        password: user.password
      })
      .then(response => {
        if (response.data.accessToken) {
          localStorage.setItem('user', JSON.stringify(response.data));
        }

        return response.data;
      });
  }

  logout() {
    localStorage.removeItem('user');
  }

  register(user) {
    return axios.post(API_URL + 'signup', {
      username: user.username,
      email: user.email,
      password: user.password
    });
  }
}

export default new AuthService();
```

For more details about ways to use Axios, please visit:
Axios request: Get/Post/Put/Delete example (https://www.bezkoder.com/axios-request/)

Data service

We also have methods for retrieving data from server. In the case we access protected resources, the HTTP request needs Authorization header.

Let’s create a helper function called `authHeader()` inside *auth-header.js*:

```
export default function authHeader() {
  let user = JSON.parse(localStorage.getItem('user'));

  if (user && user.accessToken) {
    return { Authorization: 'Bearer ' + user.accessToken };
  } else {
    return {};
  }
}
```

It checks Local Storage for `user` item.
If there is a logged in `user` with `accessToken` (JWT), return HTTP Authorization header. Otherwise, return an empty object.

Note: For Node.js Express back-end, please use **x-access-token** header like this:

```
export default function authHeader() {
  let user = JSON.parse(localStorage.getItem('user'));

  if (user && user.accessToken) {
    // for Node.js Express back-end
    return { 'x-access-token': user.accessToken };
  } else {
    return {};
  }
}
```

Now we define a service for accessing data in *user.service.js*:

```
import axios from 'axios';
import authHeader from './auth-header';

const API_URL = 'http://localhost:8080/api/test/';

class UserService {
  getPublicContent() {
    return axios.get(API_URL + 'all');
  }

  getUserBoard() {
    return axios.get(API_URL + 'user', { headers: authHeader() });
  }

  getModeratorBoard() {
    return axios.get(API_URL + 'mod', { headers: authHeader() });
  }

  getAdminBoard() {
    return axios.get(API_URL + 'admin', { headers: authHeader() });
  }
}

export default new UserService();
```

You can see that we add a HTTP header with the help of `authHeader()` function when requesting authorized resource.

Define Vuex Authentication module

We put Vuex module for authentication in **src**/**store** folder.

- store
  - auth.module.js *(authentication module)*
  - index.js *(Vuex Store that contains all modules)*

Now open *index.js* file, import `auth.module` to main Vuex Store here.

```
import { createStore } from "vuex";
import { auth } from "./auth.module";

const store = createStore({
  modules: {
    auth,
  },
});

export default store;
```

Then we start to define Vuex Authentication module that contains:

- state: { status, user }
- actions: { login, logout, register }
- mutations: { loginSuccess, loginFailure, logout, registerSuccess, registerFailure }

We use `AuthService` which is defined above to make authentication requests.

auth.module.js

```
import AuthService from '../services/auth.service';

const user = JSON.parse(localStorage.getItem('user'));
const initialState = user
  ? { status: { loggedIn: true }, user }
  : { status: { loggedIn: false }, user: null };

export const auth = {
  namespaced: true,
  state: initialState,
  actions: {
    login({ commit }, user) {
      return AuthService.login(user).then(
        user => {
          commit('loginSuccess', user);
          return Promise.resolve(user);
        },
        error => {
          commit('loginFailure');
          return Promise.reject(error);
        }
      );
    },
    logout({ commit }) {
      AuthService.logout();
      commit('logout');
    },
    register({ commit }, user) {
      return AuthService.register(user).then(
        response => {
          commit('registerSuccess');
          return Promise.resolve(response.data);
        },
        error => {
          commit('registerFailure');
          return Promise.reject(error);
        }
      );
    }
  },
  mutations: {
    loginSuccess(state, user) {
      state.status.loggedIn = true;
      state.user = user;
    },
    loginFailure(state) {
      state.status.loggedIn = false;
      state.user = null;
    },
    logout(state) {
      state.status.loggedIn = false;
      state.user = null;
    },
    registerSuccess(state) {
      state.status.loggedIn = false;
    },
    registerFailure(state) {
      state.status.loggedIn = false;
    }
  }
};
```

You can find more details about Vuex at Vuex Guide (https://vuex.vuejs.org/guide/).

Create Vue 3 Authentication Components

Let's continue with Authentication Components.
Instead of using axios or `AuthService` directly, these Components should work with Vuex Store:
– getting status with `this.$store.state.auth`
– making request by dispatching an action: `this.$store.dispatch()`

- components
  - Login.vue
  - Register.vue
  - Profile.vue

Vue 3 Login Page

In **src**/**components** folder, create *Login.vue* file with following code:

```
<template>
  <div class="col-md-12">
    <div class="card card-container">
      <img
        id="profile-img"
        src="//ssl.gstatic.com/accounts/ui/avatar_2x.png"
        class="profile-img-card"
      />
      <Form @submit="handleLogin" :validation-schema="schema">
        <div class="form-group">
          <label for="username">Username</label>
          <Field name="username" type="text" class="form-control" />
          <ErrorMessage name="username" class="error-feedback" />
        </div>
        <div class="form-group">
          <label for="password">Password</label>
          <Field name="password" type="password" class="form-control" />
          <ErrorMessage name="password" class="error-feedback" />
        </div>

        <div class="form-group">
          <button class="btn btn-primary btn-block" :disabled="loading">
            <span
              v-show="loading"
              class="spinner-border spinner-border-sm"
            ></span>
            <span>Login</span>
          </button>
        </div>

        <div class="form-group">
          <div v-if="message" class="alert alert-danger" role="alert">
            {{ message }}
          </div>
        </div>
      </Form>
    </div>
  </div>
</template>

<script>
import { Form, Field, ErrorMessage } from "vee-validate";
import * as yup from "yup";

export default {
  name: "Login",
  components: {
    Form,
    Field,
    ErrorMessage,
  },
  data() {
    const schema = yup.object().shape({
      username: yup.string().required("Username is required!"),
      password: yup.string().required("Password is required!"),
    });

    return {
      loading: false,
      message: "",
      schema,
    };
  },
  computed: {
    loggedIn() {
      return this.$store.state.auth.status.loggedIn;
    },
  },
  created() {
    if (this.loggedIn) {
      this.$router.push("/profile");
    }
  },
  methods: {
    handleLogin(user) {
      this.loading = true;

      this.$store.dispatch("auth/login", user).then(
        () => {
          this.$router.push("/profile");
        },
        (error) => {
          this.loading = false;
          this.message =
            (error.response &&
              error.response.data &&
              error.response.data.message) ||
            error.message ||
            error.toString();
        }
      );
    },
  },
};
</script>

<style scoped>
```

```
...
</style>
```

This page has a `Form` with 2 `Field` : `username & password` . We use VeeValidate 4.x (http://<a href=) to validate input. If there is an invalid field, we show the error message.

We check user logged in status using Vuex Store: `this.$store.state.auth.status.loggedIn` . If the status is `true` , we use Vue Router to direct user to **Profile Page**:

```
created() {
  if (this.loggedIn) {
    this.$router.push('/profile');
  }
},
```

In the `handleLogin()` function, we dispatch `'auth/login'` Action to Vuex Store. If the login is successful, go to **Profile Page**, otherwise, show error message.

Vue 3 Registration Page

This page is similar to **Login Page**.

For form validation, we have some more details:

- `username` : required, minimum length:3, maximum length:20
- `email` : required, email, maximum length:50
- `password` : required, minimum length:6, maximum length:40

For form submission, we dispatch `'auth/register'` Vuex Action.

components/*Register.vue*

```
<template>
  <div class="col-md-12">
    <div class="card card-container">
      <img
        id="profile-img"
        src="//ssl.gstatic.com/accounts/ui/avatar_2x.png"
        class="profile-img-card"
      />
      <Form @submit="handleRegister" :validation-schema="schema">
        <div v-if="!successful">
          <div class="form-group">
            <label for="username">Username</label>
            <Field name="username" type="text" class="form-control" />
            <ErrorMessage name="username" class="error-feedback" />
          </div>
          <div class="form-group">
            <label for="email">Email</label>
            <Field name="email" type="email" class="form-control" />
            <ErrorMessage name="email" class="error-feedback" />
          </div>
          <div class="form-group">
            <label for="password">Password</label>
            <Field name="password" type="password" class="form-control" />
            <ErrorMessage name="password" class="error-feedback" />
          </div>

          <div class="form-group">
            <button class="btn btn-primary btn-block" :disabled="loading">
              <span
                v-show="loading"
                class="spinner-border spinner-border-sm"
              ></span>
              Sign Up
            </button>
          </div>
        </div>
      </Form>

      <div
        v-if="message"
        class="alert"
        :class="successful ? 'alert-success' : 'alert-danger'"
      >
        {{ message }}
      </div>
    </div>
  </div>
</template>
```

```
<script>
import { Form, Field, ErrorMessage } from "vee-validate";
import * as yup from "yup";

export default {
  name: "Register",
  components: {
    Form,
    Field,
    ErrorMessage,
  },
  data() {
    const schema = yup.object().shape({
      username: yup
        .string()
        .required("Username is required!")
        .min(3, "Must be at least 3 characters!")
        .max(20, "Must be maximum 20 characters!"),
      email: yup
        .string()
        .required("Email is required!")
        .email("Email is invalid!")
        .max(50, "Must be maximum 50 characters!"),
      password: yup
        .string()
        .required("Password is required!")
        .min(6, "Must be at least 6 characters!")
        .max(40, "Must be maximum 40 characters!"),
    });

    return {
      successful: false,
      loading: false,
      message: "",
      schema,
    };
  },
  computed: {
    loggedIn() {
      return this.$store.state.auth.status.loggedIn;
    },
  },
  mounted() {
    if (this.loggedIn) {
      this.$router.push("/profile");
    }
  },
  methods: {
    handleRegister(user) {
      this.message = "";
      this.successful = false;
      this.loading = true;

      this.$store.dispatch("auth/register", user).then(
        (data) => {
          this.message = data.message;
          this.successful = true;
          this.loading = false;
        },
        (error) => {
          this.message =
            (error.response &&
              error.response.data &&
              error.response.data.message) ||
            error.message ||
            error.toString();
          this.successful = false;
          this.loading = false;
        }
      );
    },
  },
};
</script>

<style scoped>
...
</style>
```

Profile Page

This page gets current User from Vuex Store and show information. If the User is not logged in, it directs to **Login Page**.

components/*Profile.vue*

```
<template>
  <div class="container">
    <header class="jumbotron">
      <h3>
        <strong>{{currentUser.username}}</strong> Profile
      </h3>
    </header>
    <p>
      <strong>Token:</strong>
      {{currentUser.accessToken.substring(0, 20)}} ... {{currentUser.accessToken.substr(c
    </p>
    <p>
      <strong>Id:</strong>
      {{currentUser.id}}
    </p>
    <p>
      <strong>Email:</strong>
      {{currentUser.email}}
    </p>
    <strong>Authorities:</strong>
    <ul>
      <li v-for="role in currentUser.roles" :key="role">{{role}}</li>
    </ul>
  </div>
</template>

<script>
export default {
  name: 'Profile',
  computed: {
    currentUser() {
      return this.$store.state.auth.user;
    }
  },
  mounted() {
    if (!this.currentUser) {
      this.$router.push('/login');
    }
  }
};
</script>
```

Create Vue Components for accessing Resources

These components will use `UserService` to request data.

- components
  - Home.vue
  - BoardAdmin.vue
  - BoardModerator.vue
  - BoardUser.vue

Home Page

This is a public page.

components/*Home.vue*

```
<template>
  <div class="container">
    <header class="jumbotron">
      <h3>{{ content }}</h3>
    </header>
  </div>
</template>

<script>
import UserService from "../services/user.service";

export default {
  name: "Home",
  data() {
    return {
      content: "",
    };
  },
  mounted() {
    UserService.getPublicContent().then(
      (response) => {
        this.content = response.data;
      },
      (error) => {
        this.content =
          (error.response &&
            error.response.data &&
            error.response.data.message) ||
          error.message ||
          error.toString();
      }
    );
  },
};
</script>
```

Role-based Pages

We have 3 pages for accessing protected data:

- **BoardUser** page calls `UserService.getUserBoard()`
- **BoardModerator** page calls `UserService.getModeratorBoard()`
- **BoardAdmin** page calls `UserService.getAdminBoard()`

This is an example, other Page are similar to this Page.

components/BoardUser.vue

```
<template>
  <div class="container">
    <header class="jumbotron">
      <h3>{{ content }}</h3>
    </header>
  </div>
</template>

<script>
import UserService from "../services/user.service";

export default {
  name: "User",
  data() {
    return {
      content: "",
    };
  },
  mounted() {
    UserService.getUserBoard().then(
      (response) => {
        this.content = response.data;
      },
      (error) => {
        this.content =
          (error.response &&
            error.response.data &&
            error.response.data.message) ||
          error.message ||
          error.toString();
      }
    );
  },
};
</script>
```

Define Routes for Vue Router

Now we define all routes for our Vue 3 Application.

src/*router.js*

```
import { createWebHistory, createRouter } from "vue-router";
import Home from "./components/Home.vue";
import Login from "./components/Login.vue";
import Register from "./components/Register.vue";
// lazy-loaded
const Profile = () => import("./components/Profile.vue")
const BoardAdmin = () => import("./components/BoardAdmin.vue")
const BoardModerator = () => import("./components/BoardModerator.vue")
const BoardUser = () => import("./components/BoardUser.vue")

const routes = [
  {
    path: "/",
    name: "home",
    component: Home,
  },
  {
    path: "/home",
    component: Home,
  },
  {
    path: "/login",
    component: Login,
  },
  {
    path: "/register",
    component: Register,
  },
  {
    path: "/profile",
    name: "profile",
    // lazy-loaded
    component: Profile,
  },
  {
    path: "/admin",
    name: "admin",
    // lazy-loaded
    component: BoardAdmin,
  },
  {
    path: "/mod",
    name: "moderator",
    // lazy-loaded
    component: BoardModerator,
  },
  {
    path: "/user",
    name: "user",
    // lazy-loaded
    component: BoardUser,
  },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});

export default router;
```

Add Navigation Bar to Vue App

This is the root container for our application that contains navigation bar. We will add `router-view` here.

src/*App.vue*

```
<template>
  <div id="app">
    <nav class="navbar navbar-expand navbar-dark bg-dark">
      <a href="/" class="navbar-brand">bezKoder</a>
      <div class="navbar-nav mr-auto">
        <li class="nav-item">
          <router-link to="/home" class="nav-link">
            <font-awesome-icon icon="home" /> Home
          </router-link>
        </li>
        <li v-if="showAdminBoard" class="nav-item">
          <router-link to="/admin" class="nav-link">Admin Board</router-link>
        </li>
        <li v-if="showModeratorBoard" class="nav-item">
          <router-link to="/mod" class="nav-link">Moderator Board</router-link>
        </li>
        <li class="nav-item">
          <router-link v-if="currentUser" to="/user" class="nav-link">User</router-link>
        </li>
      </div>

      <div v-if="!currentUser" class="navbar-nav ml-auto">
        <li class="nav-item">
          <router-link to="/register" class="nav-link">
            <font-awesome-icon icon="user-plus" /> Sign Up
          </router-link>
        </li>
        <li class="nav-item">
          <router-link to="/login" class="nav-link">
            <font-awesome-icon icon="sign-in-alt" /> Login
          </router-link>
        </li>
      </div>

      <div v-if="currentUser" class="navbar-nav ml-auto">
        <li class="nav-item">
          <router-link to="/profile" class="nav-link">
            <font-awesome-icon icon="user" />
            {{ currentUser.username }}
          </router-link>
        </li>
        <li class="nav-item">
          <a class="nav-link" @click.prevent="logOut">
            <font-awesome-icon icon="sign-out-alt" /> LogOut
          </a>
        </li>
      </div>
    </nav>
```

```
      <div class="container">
        <router-view />
      </div>
    </div>
  </template>

  <script>
  export default {
    computed: {
      currentUser() {
        return this.$store.state.auth.user;
      },
      showAdminBoard() {
        if (this.currentUser && this.currentUser['roles']) {
          return this.currentUser['roles'].includes('ROLE_ADMIN');
        }

        return false;
      },
      showModeratorBoard() {
        if (this.currentUser && this.currentUser['roles']) {
          return this.currentUser['roles'].includes('ROLE_MODERATOR');
        }

        return false;
      }
    },
    methods: {
      logOut() {
        this.$store.dispatch('auth/logout');
        this.$router.push('/login');
      }
    }
  };
  </script>
```

Our navbar looks more professional when using `font-awesome-icon` .
We also make the navbar dynamically change by current User's `roles` which are retrieved from Vuex Store `state` .

Handle Unauthorized Access

If you want to check Authorized status everytime a navigating action is trigger, just add `router.beforeEach()` inside **src**/*router.js* like this:

```
router.beforeEach((to, from, next) => {
  const publicPages = ['/login', '/register', '/home'];
  const authRequired = !publicPages.includes(to.path);
  const loggedIn = localStorage.getItem('user');

  // trying to access a restricted page + not logged in
  // redirect to login page
  if (authRequired && !loggedIn) {
    next('/login');
  } else {
    next();
  }
});
```

Configure Port for Vue App

Because most of HTTP Server use CORS configuration that accepts resource sharing restricted to some sites or ports, so we also need to configure port for our App.

In project root folder, create *vue.config.js* file with following content:

```
module.exports = {
  devServer: {
    port: 8081
  }
}
```

We've set our app running at port `8081` .

Conclusion

Today we've done so many interesting things. I hope you understand the overall layers of our Vue.js application, and apply it in your project at ease. Now you can build Vue 3 app that supports JWT Authentication and Authorization with Axios, Vuex and Vue Router.

You will need following Servers for working with this Vue Client:

- Spring Boot JWT with Spring Security & MySQL (https://bezkoder.com/spring-boot-jwt-authentication/)
- Spring Boot JWT with Spring Security & PostgreSQL (https://bezkoder.com/spring-boot-security-postgresql-jwt-authentication/)
- Spring Boot JWT Authentication with Spring Security, MongoDB (https://bezkoder.com/spring-boot-jwt-auth-mongodb/)
- Node.js JWT Authentication & Authorization with MySQL (https://bezkoder.com/node-js-jwt-authentication-mysql/)
- Node.js JWT Authentication & Authorization with MongoDB (https://bezkoder.com/node-js-mongodb-auth-jwt/)

- Node.js JWT Authentication & Authorization with PostgreSQL (https://bezkoder.com/node-js-jwt-authentication-postgresql/)

Or add refresh token:
Vue 3 Refresh Token with Axios and JWT example (https://www.bezkoder.com/vue-3-refresh-token/)

Happy learning, see you again!

Further Reading

- Vue Router 4 Guide (https://next.router.vuejs.org/guide/)
- Vuex 4 Guide (https://next.vuex.vuejs.org/)
- VeeValidate 4.x (https://vee-validate.logaretm.com/v4/)
- Vue 3 CRUD example with Axios & Vue Router (https://bezkoder.com/vue-3-crud/)
- In-depth Introduction to JWT-JSON Web Token (https://bezkoder.com/jwt-json-web-token/)
- Node.js Express + Vue.js: JWT Authentication & Authorization example (https://bezkoder.com/node-express-vue-jwt-auth/)
- Spring Boot + Vue: Authentication with JWT & Spring Security Example (https://bezkoder.com/spring-boot-vue-js-authentication-jwt-spring-security/)

For more details about ways to use Axios, please visit:
Axios request: Get/Post/Put/Delete example (https://www.bezkoder.com/axios-request/)

Fullstack CRUD App:

- Vue.js + Node.js + Express + MySQL (https://bezkoder.com/vue-js-node-js-express-mysql-crud-example/)
- Vue.js + Node.js + Express + PostgreSQL (https://bezkoder.com/vue-node-express-postgresql/)
- Vue.js + Node.js + Express + MongoDB (https://bezkoder.com/vue-node-express-mongodb-mevn-crud/)
- Vue.js + Spring Boot + MySQL (https://bezkoder.com/spring-boot-vue-js-crud-example/)
- Vue.js + Spring Boot + PostgreSQL example (https://bezkoder.com/spring-boot-vue-js-postgresql/)
- Vue.js + Spring Boot + MongoDB (https://bezkoder.com/spring-boot-vue-mongodb/)
- Vue.js + Django Rest Framework (https://bezkoder.com/django-vue-js-rest-framework/)

Integration:
– Integrate Vue.js with Spring Boot (https://bezkoder.com/integrate-vue-spring-boot/)
– Integrate Vue App with Node.js Express (https://bezkoder.com/serve-vue-app-express/)

Serverless with Firebase:
– Vue Firebase Realtime Database CRUD example (https://www.bezkoder.com/vue-3-firebase/)
– Vue Firebase Firestore CRUD example (https://www.bezkoder.com/vue-3-firestore/)

Source Code

You can find the complete source code for this tutorial on Github (https://github.com/bezkoder/vue-3-authentication-jwt).

authentication (https://www.bezkoder.com/tag/authentication/)

authorization (https://www.bezkoder.com/tag/authorization/) jwt (https://www.bezkoder.com/tag/jwt/)

registration (https://www.bezkoder.com/tag/registration/) security (https://www.bezkoder.com/tag/security/)

token based authentication (https://www.bezkoder.com/tag/token-based-authentication/)

vue (https://www.bezkoder.com/tag/vue/) vue 3 (https://www.bezkoder.com/tag/vue-3/)

vue router (https://www.bezkoder.com/tag/vue-router/) vuex (https://www.bezkoder.com/tag/vuex/)

14 thoughts to "Vue 3 Authentication with JWT, Vuex, Axios and Vue Router"

Vu Khoi
June 5, 2021 at 2:56 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-8883)

Thanks for best tutorial

REPLY

Lahiru Lomitha
June 8, 2021 at 4:05 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-8942)

This is very useful content thank you for sharing it with us. Also is there a tutorial about using refresh tokens in the Vue3 application. Also, using HTTP only cookies to store JWT.

REPLY

Walt
June 15, 2021 at 5:49 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-9078)

Many thanks for this excellent tutorial. Would be nice to see a similar example using the Vue 3 Composition API.

REPLY

Emanoel Evaristo de Sousa
July 22, 2021 at 8:25 pm (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-9868)

Yes, Would be very nice

REPLY

Jihad
August 14, 2021 at 7:02 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-10814)

Exactly!

REPLY

moragreen
July 27, 2021 at 7:15 pm (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-10130)

Awesome Vue3 tutorial. Extremely comprehensive and easy to understand. Thanks!!!

REPLY

Clarissa
August 5, 2021 at 11:40 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-10449)

This tutorial is well-written. Thanks!

REPLY

svenchristmas
August 7, 2021 at 2:21 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-10503)

Perfect tutorial, it is really cool!

REPLY

arnett
August 8, 2021 at 3:55 pm (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-10566)

Brief but very accurate tutorial. Many thanks for sharing this one!
A must read post!

REPLY

Jihad
August 14, 2021 at 7:00 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-10813)

Thank you very much for this great tutorial!
Very clear and well described, However I have one concern, do you think that importing UserService directly into Home.vue component is the best practice?

Wouldn't it be better if we let Vuex to interact with the UserService as we did for the AuthService?

I mean creating a user.js module in store same we as we create auth.js and then from Home.vue (or any other component) to dispatch to the actions in user.js module.

REPLY

bezkoder
August 16, 2021 at 2:03 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-10887)

Hi, `UserService` (in this tutorial, just like a data service) is an example which indicates that we don't need to put everything in the store 🙂
Generally we use Vuex store when we need the same data at two completely different places in our application. This is the point Vuex makes a lot of sense.

REPLY

filome
August 26, 2021 at 5:26 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-11233)

This article provides clear idea for the new programmers! Thanks!

REPLY

frederick
August 30, 2021 at 3:48 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-11400)

Perfect work you have done, this tutorial is really cool with good explanation.

REPLY

delhi escorts
September 12, 2021 at 8:49 am (https://www.bezkoder.com/vue-3-authentication-jwt/#comment-12062)

Thank you and best of luck.

REPLY

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

☐

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

< Vue 3 Typescript example with Axios: Build CRUD App (https://www.bezkoder.com/vue-3-typescript-axios/)

[Full-stack] Spring Boot + Vue.js: CRUD example > (https://www.bezkoder.com/spring-boot-vue-js-crud-example/)

Search...

FOLLOW US

(https://facebook.com/bezkoder)

(https://www.youtube.com/channel/UCp0mx9RH0JxaFsmMvKXA86Q)

(https://github.com/bezkoder)

TOOLS

Json Formatter (https://www.bezkoder.com/json-formatter/)

Home (https://bezkoder.com/) Privacy Policy (https://www.bezkoder.com/privacy-policy/)

Contact Us (https://www.bezkoder.com/contact-us/) About Us (https://www.bezkoder.com/about/)