

Using Vue with ASP.NET Core By Example

Lab 8 Instructions

Shawn Wildermuth, Instructor

1. Adding a Route Guard
    a. Open the **app.js** file.
    b. Find the creation of the **router** in the Vue object.
    c. Separate it out into its own variable and use that variable in the **Vue** object:

```
let theRouter = new VueRouter({ routes });

let vm = new Vue({
  el: "#index-view",
  store,
  components: {
    Cart
  },
  router: theRouter
});
```

    d. After creation of the **theRouter**, call **beforeEach** with a function callback to guard the routes.
    e. Call **next()** inside the callback:

```
let theRouter = new VueRouter({ routes });

theRouter.beforeEach(function (to, _from, next) {
  next();
});
```

    f. Add a **meta** property to the **checkout** route and set **authRequired** to **true** inside of it:

```
let routes = [
  { path: "/", component: ProductList },
  {
    path: "/checkout",
    component: Checkout,
    meta: {
      authRequired: true
    }
  }
];
```

    g. Open the **store.js** file.
    h. Add properties for **token** and **expiration** to the list of state:

```
export default new Vuex.Store({
  state: {
    cart: [],
    products: [],
    token: "",
    expiration: new Date()
  },
```

i. Add a new getter called **isAuthenticated**:

```
  getters: {
    isAuthenticated: (state) => state.token && state.expiration > new Date()
  }
```

j. Open the **app.js** again.

k. In the route guard, check the **isAuthenticated** and call **next** with a login view (which we haven't created yet):

```
theRouter.beforeEach(function (to, _from, next) {
  if (to.meta.authRequired) {
    if (store.getters.isAuthenticated == false) {
      next({ path: "/login" });
      return; // Prevent the main next() from being called
    }
  }
  next();
});
```

l. Run the app and try to go to checkout and see it go to /login (though the page doesn't work).

2. Creating a Login View
    a. Create a new file in the **js** folder called **login.js**
    b. Import the Vue object.
    c. Export a new Vue component:

```
// Login.js
import Vue from "vue";

export default Vue.component("login", {

})
```

    d. Create an empty template property.
    e. Create a data callback that returns a credential object and an empty error string:

```
// Login.js
import Vue from "vue";

export default Vue.component("login", {
  template: ``,
  data() {
    return {
      credentials: {},
      error: ""
    }
  }
})
```

f. Create the template (make sure you're using the back single quote to surround it as shown above):

```
<div class="col-md-6 offset-3">
  <h3>Login</h3>
  <div v-if="error" class="text-danger">{{ error }}</div>
  <form novalidate>
    <div class="form-group">
      <label for="username">Username</label>
      <input type="text"
             name="username"
             class="form-control"
             v-model="credentials.username" />
    </div>
    <div class="form-group">
      <label for="password">Password</label>
      <input type="password"
             name="password"
             class="form-control"
             v-model="credentials.password" />
    </div>
    <div class="form-group">
      <input type="submit" class="btn btn-success" value="Login" />
    </div>
  </form>
</div>
```

g. Open the app.js again.
h. Import the Login component:

```
// app.js
import Vue from "vue";
import VueRouter from "vue-router";
import VeeValidate from "vee-validate";

import ProductList from "./productList";
import Checkout from "./checkout";
import store from "./store";
import Cart from "./Cart";
import Login from "./login";
```

i. Add a new route for the login component.

```
let routes = [
  { path: "/", component: ProductList },
  {
    path: "/checkout",
    component: Checkout,
    meta: {
      authRequired: true
    }
  },
  {
    path: "/login",
    component: Login
  }
];
```

j. Run the project and see the login page shown.

3. Getting a Token
    a. Open the **login.js.**
    b. Add a handler for **submit** of the **form** and call **onSubmit**:

```
export default Vue.component("login", {
  template: `
<div class="col-md-6 offset-3">
  <h3>Login</h3>
  <div v-if="error" class="text-danger">{{ error }}</div>
  <form novalidate @submit.prevent="onSubmit">
```

    c. Add a section for **methods**:

```
data() {
    return {
      credentials: {},
      error: ""
    }
  },
  methods: {
  }
})
```

    d. Create a new **onSubmit** method.
    e. Inside, call the **$store's dispatch** method to call an **action** on our **Vuex store** called **onAuth**.
    f. Pass in the **credentials** to the **dispatch**.
    g. Call **then** and **catch** as well:

```
    onSubmit() {
      this.$store.dispatch("doAuth", this.credentials)
        .then()
        .catch(() => this.error = "Failed to login");
    }
```

    h. In the then, use the router to redirect to the checkout:

```
    onSubmit() {
      this.$store.dispatch("doAuth", this.credentials)
        .then(() => {
          this.$router.push("/checkout");
        })
        .catch(() => this.error = "Failed to login");
    }
```

i. Open the **store.js** file.
j. Create a new **action** called **doAuth**.
k. Set the **context** and **creds** as parameters:

```
doAuth(context, creds) {
}
```

l. Create and return a **Promise** like we've done before:

```
doAuth(context, creds) {
  return new Promise(function (resolve, reject) {

  });
}
```

m. Inside the **promise**, use **axios** to **post** the credentials to the API for creating the token.
n. In the **then** and **catch**, use **resolve** and **reject** like we've done before:

```
doAuth(context, creds) {
  return new Promise(function (resolve, reject) {
    axios.post("/api/account/createtoken", creds)
      .then((res) => {
        resolve();
      })
      .catch(() => reject());
  });
}
```

o. Inside the **then**, use the **context** to **commit** a new **mutation** called **setToken**:

```
doAuth(context, creds) {
  return new Promise(function (resolve, reject) {
    axios.post("/api/account/createtoken", creds)
      .then((res) => {
        context.commit("setToken", res.data);
        resolve();
      })
      .catch(() => reject());
  });
}
```

p. Create a new **mutation** called **setToken** with **state** and **tokenInfo** as parameters:

```
mutations: {
  setProducts: function (state, newCatalog) {
    state.products = newCatalog;
  },
  addProductToCart: function(state, product) {
    state.cart.push(product);
  },
  clearCart: function (state) {
    state.cart = [];
  },
  setToken(state, tokenInfo) {

  }
},
```

q. Set the token and expiration using the token info:

```
  setToken(state, tokenInfo) {
    state.token = tokenInfo.token;
    state.expiration = new Date(tokenInfo.expiration);
  }
```

r. Finally, run the project and click on checkout to redirect to login.

s. Then once you fill out the username and password (shawn@wildermuth.com and P@ssw0rd! by default), it'll take you to the checkout page.

t. Cancel the checkout page and click checkout again and since you're already authenticated, it'll skip the login page.

4. Using the Token
    a. Open the **CustomerController.cs** in the **Controllers** directory.
    b. Change the authentication to use **JwtBearer** only:

```
namespace TheStore.Controllers.Api
{
  [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
  [Route("api/[controller]")]
  public class CustomerController : Controller
```

    c. Open the **checkout.js** file.
    d. Import **axios** into the top of the file:

```
// checkout.js
import Vue from "vue";
import axios from "axios";

export default Vue.component("checkout", {
```

    e. In the **data** section, add a member for "**msg**":

```
  data: function () {
    return {
      customer: {},
      msg: ""
    };
  },
```

    f. At the end of the **template** show this new **msg**:

```
    <div class="form-group">
      <router-link to="/" class="btn btn-info">Cancel</router-link>
      <input type="submit" class="btn" value="Save" />
    </div>
    <div>{{ msg }}</div>
  </form>
</div>`,
```

g. Replace the call to **alert** in the **onSave** with a **post** to the server to **/api/customer**:

```
methods: {
  onSave: function () {
    axios.post("/api/customer", this.customer)
      .then(() => this.msg = "Saved....")
      .catch((err) => this.msg = err.response.statusText + ":" + err.response.data)
  }
}
```

h. Notice how you're using the **msg** to show the result.

i. Run this and see it fail.

j. Finally, add the **authorization header** as a third parameter in the call to **post**:

```
methods: {
  onSave: function () {
    axios.post("/api/customer", this.customer, {
      headers: {
        "Authorization": "Bearer " + this.$store.state.token
      }
    })
      .then(() => this.msg = "Saved....")
      .catch((err) => this.msg = err.response.statusText + ":" + err.response.data)
  }
}
```

k. If you try it again, you'll see it work. Subsequent calls will fail as it's only allowed to be posted once.