Using Vue with ASP.NET Core By Example

Lab 5 Instructions

Shawn Wildermuth, Instructor

1. Adding Vuex to the Project
    a. Open the **package.json** file.
    b. Add "**vuex**" to the list of packages:

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "dependencies": {
    "jquery": "~3.3.1",
    "popper.js": "1.14.1",
    "bootstrap": "4.0.0",
    "bootswatch": "4.0.0",
    "@fortawesome/fontawesome-free-webfonts": "1.0.5",
    "vue": "2.5.16",
    "axios": "0.18.0",
    "vee-validate": "2.0.8",
    "vuex": "3.0.1"
  }
}
```

    c. Once Visual Studio has updated the packages, open the **index.cshtml** file and add **vuex** to the list of scripts:

```
@section scripts {
  <script src="~/node_modules/vue/dist/vue.js"></script>
  <script src="~/node_modules/axios/dist/axios.js"></script>
  <script src="~/node_modules/vuex/dist/vuex.js"></script>
  <script src="~/js/cart.js"></script>
  <script src="~/js/index.js"></script>
}
```

    d. Create a new file in **wwwroot/js** called "**store.js**".
    e. Drag this new file to the scripts section of **index.cshtml** to add it to the list of scripts:

```
@section scripts {
  <script src="~/node_modules/vue/dist/vue.js"></script>
  <script src="~/node_modules/axios/dist/axios.js"></script>
  <script src="~/node_modules/vuex/dist/vuex.js"></script>
  <script src="~/js/store.js"></script>
  <script src="~/js/cart.js"></script>
  <script src="~/js/index.js"></script>
}
```

f. In the new file, instantiate the new **Vuex.Store** with properties for **state**, **mutations**, **actions** and **getters**:

```
// store.js
let store = new Vuex.Store({
  state: {},
  mutations: {},
  actions: {},
  getters: {}
});
```

g. Open the **index.js**.

h. Add **store** as a new property in the **Vue** object:

```
// index.js
new Vue({
  el: "#index-view",
  store,
  data: {
    appName: "Product List",
    cart: [],
    products: [],
    error: ""
  },
```

2. Sharing Data in the Store
    a. Open the **index.js** file.
    b. Cut the **catalog** and **cart** properties out of the main **Vue** object.
    c. Open the **store.js** and paste the two properties into the **state** object:

```
// store.js
let store = new Vuex.Store({
  state: {
    cart: [],
    product: [],
  },
  mutations: {},
  actions: {},
  getters: {}
});
```

    d. Next in the **index.js**, cut the call to **axios** to fill the catalog and create a new function in actions in **store.js**:

```
  actions: {
    loadCatalog: function (context) {
      axios.get("/api/products")
        .then(function (res) {
          this.products = res.data.results;
        }.bind(this))
        .catch(function () {
          this.error = "Failed to load products";
        }.bind(this));
    }
  },
```

    e. In **loadCatalog**, wrap the call to **axios** with a return of a **Promise** object:

```
  actions: {
    return new Promise(function (resolve, reject) {
      loadCatalog: function (context) {
        axios.get("/api/products")
          .then(function (res) {
            this.products = res.data.results;
          }.bind(this))
          .catch(function () {
            this.error = "Failed to load products";
          }.bind(this));
      }
    });
  },
```

f. Create a new **mutation** called **setProducts** taking in state and **newCatalog**:

```
mutations: {
  setProducts: function (state, newCatalog) {
    state.products = newCatalog;
  }
},
```

g. Change the call to loadProducts to get rid of calls to bind, and use the mutation to set the products:

```
  loadCatalog: function (context) {
    return new Promise(function (resolve, reject) {
      axios.get("/api/products")
        .then(function (res) {
          context.commit("setProducts", res.data.results);
        })
        .catch(function () {
          this.error = "Failed to load products";
        });
    });
  }
```

h. Call **resolve** and **reject** to complete the **Promise** object:

```
  loadCatalog: function (context) {
    return new Promise(function (resolve, reject) {
      axios.get("/api/products")
        .then(function (res) {
          context.commit("setProducts", res.data.results);
          resolve();
        })
        .catch(function () {
          reject();
        });
    });
```

i. Now we're ready to bind to the products.

3. Binding to Vuex
    a. Open the **index.cshtml** file.
    b. Change the **v-for** to use the **store**:

```
  <div class="row">
    <div class="col-md-8">
      <div v-for="p in $store.state.products">
        <div>{{ p.name }}</div>
        <div>{{ p.description }}</div>
        <div>{{ p.listPrice }}</div>
        <button class="btn btn-info" v-on:click="onBuy(p)">Buy</button>
        <hr />
      </div>
    </div>
    <div class="col-md-4">
      <the-cart v-bind:items="$store.state.cart" v-on:empty-cart="onEmptyCart()"></the-
cart>
    </div>
  </div>
```

    c. Change the call to use the **cart.length** to use the **store's** cart:

```
  <div>Purchased: {{ $store.state.cart.length }}</div>
```

    d. Run the project to see if it still works (products won't show yet).
    e. In the **index.js**, change the code in the **mounted** to use the **action** (you don't need then since you're not doing anything with the return data):

```
  mounted: function () {
    this.error = "";
    this.$store.dispatch("loadCatalog")
      .catch(function () { this.error = "Could not load catalog." });
  },
```

    f. Run again to see the products.

4. Implement the Cart
    a. In the **index.js**, change the **onBuy** to call a new **mutation** called "**addProductToCart**":

```
  onBuy: function (product) {
    this.$store.commit("addProductToCart", product);
  },
```

    b. Open the **store.js**, and **push** the new **product** to the cart:

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

    c. Run to see if you can add items to the cart.
    d. In the index.js, change the onEmptyCart to call the store:

```
  onEmptyCart: function () {
    this.$store.commit("clearCart");
  }
```

    e. In the **store.js**, add the new **mutation** to clear the cart:

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

    f. Run and see the cart working as well.