Actions

NEXT: STATE AND ACTIONS ON COMPONENTS

Actions work alongside Mutations. They are used to handle asynchronous tasks which mutations can't do.

The flow is -- your component *dispatches* an action, the action makes and async request and *commits* to mutation. Mutations update the state and state are *passed* to components to update the view where necessary.

We have been showing how to commit to mutations from the component. From the flow described, we need to move that to actions and start dispatching actions from the component.

Let's create our actions.

First we still need to import the mutation constants and `axios` for HTTP requests:

```JS
// ./src/store/actions
import axios from 'axios'
const API_BASE = '<API-URL/api/v1>'

import {
  ADD_PRODUCT,
  ADD_PRODUCT_SUCCESS,
  PRODUCT_BY_ID,
  PRODUCT_BY_ID_SUCCESS,
  UPDATE_PRODUCT,
  UPDATE_PRODUCT_SUCCESS,
  REMOVE_PRODUCT,
  REMOVE_PRODUCT_SUCCESS,
  ALL_PRODUCTS,
  ALL_PRODUCTS_SUCCESS,
  ALL_MANUFACTURERS,
  ALL_MANUFACTURERS_SUCCESS
} from './mutation-types'
```

The actions which is what are actually triggered in our components and in turn trigger mutations are simple. The first which is `allProducts` ,commits the `ALL_PRODUCTS` mutation to start a loading spinner. It then makes a HTTP request with `axios` , and commits a `SUCCESS` mutation with the returned data:

```JS
export const productActions = {
  allProducts ({commit}) {
    commit(ALL_PRODUCTS)
    // Fetch actual products from the API
    axios.get(`${API_BASE}/products`).then(response => {
      commit(ALL_PRODUCTS_SUCCESS, response.data)
    })
  },
```

Same pattern is applied to the rest of the actions as shown below:

```JS
productById ({commit}, payload) {
    commit(PRODUCT_BY_ID)
    // Fetch product by ID from API
    axios.get(`${API_BASE}/products/${payload}`).then(response => {
      commit(PRODUCT_BY_ID_SUCCESS, response.data)
    })
  },
  addProduct ({commit}, payload) {
    commit(ADD_PRODUCT)
    // Create a new product via API
    axios.post(`${API_BASE}/products`, payload).then(response => {
      commit(ADD_PRODUCT_SUCCESS, response.data)
    })
  },
  updateProduct ({commit}, payload) {
    commit(UPDATE_PRODUCT)
    // Update product via API
    axios.put(`${API_BASE}/products/${payload._id}`, payload).then(response => {
      commit(UPDATE_PRODUCT_SUCCESS, response.data)
    })
  },
  removeProduct ({commit}, payload) {
    commit(REMOVE_PRODUCT)
    // Delete product via API
    axios.delete(`${API_BASE}/products/${payload}`, payload).then(response => {
      console.debug('response', response.data)
      commit(REMOVE_PRODUCT_SUCCESS, response.data)
    })
  }
}
```

Cart does not require actions as it's just a UI thing. Listing manufacturers on the other hand relies on async operations so an action is needed:

```JS
export const manufacturerActions = {
  allManufacturers ({commit}) {
    commit(ALL_MANUFACTURERS)
    // Fetch all manufacturers from API
    axios.get(`${API_BASE}/manufacturers`).then(response => {
      commit(ALL_MANUFACTURERS_SUCCESS, response.data)
    })
  }
}
```

Notice how each action commits to our mutations using the mutation types. The actions are asynchronous. The commit to a mutation that first shows the loading the spinner before starting a request to the server with `axios` . When a response is returned, it commits to another mutation to hide the loading spinner and update the state.

Remember all components tied to state when the state is updated, gets updated as well.