Mutations

NEXT: ACTIONS

You might be tempted to update state from your component this way:

JS

```
this.products.push({})
```

Vue needs to keep track of these changes when they are made. Therefore, the above is not allowed. Changes are made to the state using a store member, **Mutations**.

When a change needs to be made, we commit to mutations, and the mutation function makes this update. Mutations must by synchronous, Actions are for asynchronous tasks.

Table of Contents

JS

```
export const productMutations = {
  addProduct (state, payload) {
    state.showLoader = true
    state.products.push(payload)
  }
}
//
//
// store
mutations: Object.assign({}, productMutations)
//
//
// commit
methods: {
  addProduct(product) {
    this.$store.commit('addProduct', product)
  }
}
```

The `commit` method takes the name of the mutation handler and the payload. The handler `addProduct` receives the current state and payload and appends the payload to the state's products.

Constants as Mutation Types

`addProduct` must be used in two or more places. When creating the mutation and while committing to the mutation. This creates a layer of possible typographical error. You can use constants to replace these types and allow tools like IntelliSense to help you avoid typos and errors:

JS

```
// ./src/store/mutation-types
export const ALL_PRODUCTS = 'ALL_PRODUCTS'
export const ALL_PRODUCTS_SUCCESS = 'ALL_PRODUCTS_SUCCESS'

export const PRODUCT_BY_ID = 'PRODUCT_BY_ID'
export const PRODUCT_BY_ID_SUCCESS = 'PRODUCT_BY_ID_SUCCESS'

export const ADD_PRODUCT = 'ADD_PRODUCT'
export const ADD_PRODUCT_SUCCESS = 'ADD_PRODUCT_SUCCESS'

export const UPDATE_PRODUCT = 'UPDATE_PRODUCT'
export const UPDATE_PRODUCT_SUCCESS = 'UPDATE_PRODUCT_SUCCESS'

export const REMOVE_PRODUCT = 'REMOVE_PRODUCT'
export const REMOVE_PRODUCT_SUCCESS = 'REMOVE_PRODUCT_SUCCESS'

export const ADD_TO_CART = 'ADD_TO_CART'
export const REMOVE_FROM_CART = 'REMOVE_FROM_CART'

export const ALL_MANUFACTURERS = 'ALL_MANUFACTURER'
export const ALL_MANUFACTURERS_SUCCESS = 'ALL_MANUFACTURER_SUCCESS'
```

Just a string mapped to a constant. Nothing more.

We can now use these constants both for creating mutations and committing to them.

First import the constants into our mutations file:

JS

```
// ./src/store/mutations

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

Now we have access to the them and can use them as mutation function names. The most important mutation is the `product` mutation which mutates the product state. Here are the mutation methods:

JS

```
// ./src/store/mutations

// Constant imports ...

export const productMutations = {
  [ALL_PRODUCTS] (state) {
    // Called when fetching products
    state.showLoader = true
  },
  [ALL_PRODUCTS_SUCCESS] (state, payload) {
    // Called when products have been fetched
    state.showLoader = false
    // Updates state products
    state.products = payload
  },
  [PRODUCT_BY_ID] (state) {
    // Called when fetching products by ID
    state.showLoader = true
  },
  [PRODUCT_BY_ID_SUCCESS] (state, payload) {
    // Called when product is fetched
    state.showLoader = false
    // Updates state product
    state.product = payload
  },
  [ADD_PRODUCT]: (state, payload) => {
    // ...Same pattern
    state.showLoader = true
  },
  [ADD_PRODUCT_SUCCESS]: (state, payload) => {
    state.showLoader = false
    state.products.push(payload)
  },
  [UPDATE_PRODUCT]: (state, payload) => {
    state.showLoader = true
  },
  [UPDATE_PRODUCT_SUCCESS]: (state, payload) => {
    state.showLoader = false
    state.products = state.products.map(p => {
      if (p._id === payload._id) {
        payload = {...payload, manufacturer: state.manufacturers.filter(x => x._id === payload.man
        return payload
      }
      return p
    })
  },
  [REMOVE_PRODUCT]: (state, payload) => {
    state.showLoader = true
  },
  [REMOVE_PRODUCT_SUCCESS]: (state, payload) => {
    state.showLoader = false
    const index = state.products.findIndex(p => p._id === payload)
    console.debug('index', index)
    state.products.splice(index, 1)
  }
}
```

Each of the mutation has a `SUCCESS` variation which is called after mutation is completed. The actual mutations just kicks of a pending state by showing the loading spinner. Then the `SUCCESS` mutations update the UI.

Next, we do the same thing for the cart and manufacturer:

JS

```
// ./src/store/mutations
export const cartMutations = {
  [ADD_TO_CART]: (state, payload) => state.cart.push(payload),
  [REMOVE_FROM_CART]: (state, payload) => {
    const index = state.cart.findIndex(p => p._id === payload)
    state.cart.splice(index, 1)
    console.log(state.cart, state.cart.length, index)
  }
}

export const manufacturerMutations = {
  [ALL_MANUFACTURERS] (state) {
    state.showLoader = true
  },
  [ALL_MANUFACTURERS_SUCCESS] (state, payload) {
    state.showLoader = false
    state.manufacturers = payload
  }
}
```

The mutations are added to the store afterwards:

JS

```
mutations: Object.assign({}, productMutations, cartMutations, manufacturerMutations),
```

Next, we take a look at how `actions` trigger these mutations for for async operations.

Getting Started

Vue Basics

Routing

Forms

API Backend

Vuex

Using Store In Components