

Vuex. Modules

Для чого потрібні модулі?

У випадку, коли у додатку використовуються різні сутності.

Наприклад, зберігаються дані:

- про користувачів,
- про товари
- про виробників
- про постачальників
- про замовлення
-



Для чого потрібні модулі?

У випадку, коли у додатку використовуються різні сутності то :

1) у state доведеть зберігати дані різних сутностей



2) у getters доводиться зберігти геттери для роботи з різними сутностями

3) у mutations мутації для роботи з різними сутностями

.....

Для чого потрібні модулі?

У випадку, коли у додатку використовуються різні сутності то :

1) у state доведеть зберігати дані різних сутностей

2) у getters доводиться зберігти геттери для роботи з різними сутностями

3) у mutations мутації для роботи з різними сутностями

.....

Легко заплутатись до яких сутностей відносяться методи для роботи з даними

```
const store = createStore({
  state: () => {
    return {
      products: [ ... ],
      users: [ ... ],
    }
  },
  getters: {
    products: ({ products }) => products,
    users: ({ users }) => users,
    getUserProductsData: (state) => (userId) => {
      ...
    },
  },
  mutations : {
    editProduct(state, editedProduct) { ... },
    deleteProduct(state, productId) { ... },
    addUser(state, user) { ... },
    editUser(state, editedUser) { ... },
    deleteUser(state, userId) { ... },
  },
},
```



Модуль 1

```
const moduleA = {  
  state: () => ({ ... }),  
  mutations: { ... },  
  actions: { ... },  
  getters: { ... }  
}
```

Модуль 2

```
const moduleB = {  
  state: () => ({ ... }),  
  mutations: { ... },  
  actions: { ... }  
}
```

1) Обробку даних окремих сутностей виносimo в окремi модулi



Модуль 1

```
const moduleA = {  
  state: () => ({ ... }),  
  mutations: { ... },  
  actions: { ... },  
  getters: { ... }  
}
```

Модуль 2

```
const moduleB = {  
  state: () => ({ ... }),  
  mutations: { ... },  
  actions: { ... }  
}
```

1) Обробку даних окремих сутностей виносимо в окремі модули

2) Створення кореневого сховища, яке об'єднує модули

```
const store = createStore({  
  state() { . . . . . },  
  getters: { . . . . . },  
  mutations: { . . . . . },  
  actions: { . . . . . },  
  
  //---підключення модулів ---  
  modules:{  
    модуль1,  
    модуль2,  
    . . . . .  
    нова_назва_модуля : модуль  
    . . . . .  
  }  
})
```

```
const store = createStore({  
  modules: {  
    a: moduleA,  
    b: moduleB  
  }  
})
```

store.state.a // -> `moduleA`'s state
store.state.b // -> `moduleB`'s state



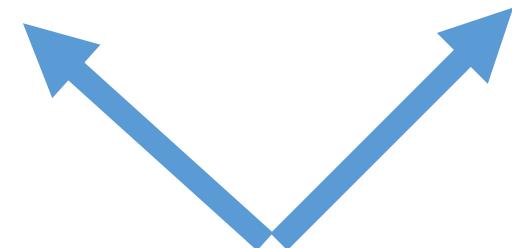
js products.js store U X

```
src > store > JS products.js > [o] getters > ⚡ products
1 const state = {
2   products: [
3     { id: 1, title: 'Продукт 1', price: 100 },
4     { id: 2, title: 'Продукт 2', price: 200 },
5     { id: 3, title: 'Продукт 3', price: 150 },
6     { id: 4, title: 'Продукт 4', price: 120 },
7     { id: 5, title: 'Продукт 5', price: 180 },
8   ],
9 }
10 const getters = {
11   products: ({ products }) => products,
12 }
13 > const mutations = ...
14
15 > const actions = ...
16
17 export default {
18   namespaced: true,
19   state,
20   getters,
21   mutations,
22   actions,
23 }
```

src > store > JS users.js > [o] getters

```
1 const state = {
2   users: [
3     { id: 1, name: 'Користувач 1', productsList: [1, 3] },
4     { id: 2, name: 'Користувач 2', productsList: [2, 4] },
5     { id: 3, name: 'Користувач 3', productsList: [1, 5] },
6     { id: 4, name: 'Користувач 4', productsList: [2, 3, 4] },
7     { id: 5, name: 'Користувач 5', productsList: [5] },
8   ],
9 }
10 > const getters = ...
11 }
12
13 > const mutations = ...
14
15 > const actions = ...
16
17 export default {
18   namespaced: true,
19   state,
20   getters,
21   mutations,
22   actions,
23 }
```

Приклад. Сховище складається з
двох частин:
1) для роботи з списком товарів
2) для роботи з списком
користувачів



1) Обробку даних окремих сутностей виносimo в окремi модулi

src

- > assets
- > components
- < store
 - JS index.js M
 - JS products.js U
 - JS users.js U
- < App.vue M
- JS main.js

```
src > store > JS products.js > [?] getters > ? products
 1  const state = {
 2    products: [
 3      { id: 1, title: 'Продукт 1', price: 100 },
 4      { id: 2, title: 'Продукт 2', price: 200 },
 5      { id: 3, title: 'Продукт 3', price: 150 },
 6      { id: 4, title: 'Продукт 4', price: 120 },
 7      { id: 5, title: 'Продукт 5', price: 180 },
 8    ],
 9  }
10 const getters = {
11   products: ({ products }) => products,
12 }
13 > const mutations = { ... }
14
15 > const actions = { ... }
16
17
18 export default {
19   namespaced: true,
20   state,
21   getters,
22   mutations,
23   actions,
24 }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40 }
```

Приклад. Сховище складається з двох частин:

- 1)для роботи з списком товарів
- 2)для роботи з списком користувачів

1) Обробку даних окремих сутностей виносимо в окремі модулі

A screenshot of a file explorer interface. The tree view shows the following structure:

- src
 - assets
 - components
 - store
 - index.js (highlighted with a blue border)
 - products.js (highlighted with a green border)
 - users.js (highlighted with a red border)
- App.vue
- main.js

Each file or folder has a status indicator to its right: M for modified (index.js), U for untracked (products.js, users.js), and a grey dot for others.

The screenshot shows a code editor with a file named `index.js` open. The code defines a `store` using the `createStore` function from `'vuex'`. It imports two modules: `productsModule` and `usersModule`. The `store` instance is created with `modules` containing `products` and `users` properties pointing to their respective modules.

```
src > store > JS index.js > ...
You, 3 seconds ago | 1 author (You)
1 import { createStore } from 'vuex'
2 import productsModule from './products'
3 import usersModule from './users'
4
5 // Create a new store instance.
6 const store = createStore({
7   ...modules: {
8     ...products: productsModule,
9     ...users: usersModule,
10   },
11 }
12 export default store
```

src > store > **JS** users.js > [e] getters

```
1 const state = {
2   users: [
3     { id: 1, name: 'Користувач 1', productsList: [1, 3] },
4     { id: 2, name: 'Користувач 2', productsList: [2, 4] },
5     { id: 3, name: 'Користувач 3', productsList: [1, 5] },
6     { id: 4, name: 'Користувач 4', productsList: [2, 3, 4] },
7     { id: 5, name: 'Користувач 5', productsList: [5] },
8   ],
9 }
10
11 > const getters = { ... }
12
13 > const mutations = { ... }
14
15 > const actions = { ... }
16
17
18 export default {
19   namespaced: true,
20   state,
21   getters,
22   mutations,
23   actions,
24 }
```

2)Створення кореневого сховища

Використання геттерів з модулів у компонентах

Загальна схема

```
<template>
```

```
</template>
```

```
<script>
```

```
//1. Імпортуємо mapGetters
```

```
import { mapGetters } from 'vuex'
```

```
export default {
```

```
}
```

```
</script>
```

Приклад



The screenshot shows the code editor interface with the file 'App.vue' open. The code is as follows:

```
<template>
  ...
</template>

<script>
  import { mapGetters } from 'vuex'
  ...
  export default {
    name: 'App',
    ...
    data() {
      ...
    },
    ...
    computed: {
      ...
    },
    ...
    methods: {}
  }
</script>
```

A red rectangular box highlights the line 'import { mapGetters } from 'vuex''.

Використання геттерів з модулів у компонентах

Загальна схема

```
<template>
```

```
</template>
```

```
<script>
```

//1. Імпортуюмо *mapGetters*

```
import { mapGetters } from 'vuex'
```

```
export default {
```

```
    . . . . . . . . .
```

//2. Підключаємо геттери

```
computed: {
    ...mapGetters('назва_модуля', ['геттер1', ...]),
},
```

```
    . . . . . . . . .
```

```
}
```

```
</script>
```

Приклад

V App.vue src M X

src > V App.vue > {} "App.vue" > script

You, 2 minutes ago | 1 author (You)

```
1 <template>
```

```
2 . . .
```

```
3 . . .
```

```
4 . . .
```

```
5 . . .
```

```
6 . . .
```

```
7 </template>
```

```
8
```

```
9 <script>
```

```
10 import { mapGetters } from 'vuex'
```

```
11
```

```
12 export default {
```

```
13     name: 'App',
```

```
14
```

```
15     components: {},
```

```
16
```

```
17     data() {
```

```
18         . . . return {}
```

```
19     },
```

```
20
```

```
21     computed: {
```

```
22         . . . mapGetters('users', ['users', 'getUserProductsData'])
```

```
23     },
```

```
24
```

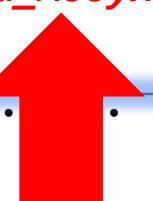
```
25     methods: {},
```

```
26 }
```

```
27 </script>
```

```
28
```

```
29
```



Використання геттерів з модулів у компонентах

Загальна схема

```
<template>
//3. Використовуємо геттери як звичайну
обчислювальну властивість
    . . . . .
    <тег>{{ геттер }}</тег >
    . . . . .
    <тег :атрибут = “геттер” />
    . . . . .

</template>
<script>
//1. Імпортуюмо mapGetters
import { mapGetters } from 'vuex'

export default {
    . . . . .
}

//2. Підключаємо геттери
computed: {
    ...mapGetters('назва_модуля', ['геттер1', ...]),
},
    . . . . .

}
</script>
```

Приклад

The screenshot shows a code editor with a dark theme. The file is named 'App.vue' and is located in the 'src' directory. The code consists of a template section and a script section.

```
App.vue src M X
src > App.vue > {} "App.vue" > script
You, 2 minutes ago | 1 author (You)
1 <template>
2     <div>{{ users }}</div>
3     <hr />
4     <div>{{ getUserProductsData(1) }}</div>
5
6
7 </template>
8
9 <script>
10 import { mapGetters } from 'vuex'
11
12 export default {
13     name: 'App',
14
15     components: {},
16
17     data() {
18         return {}
19     },
20
21     computed: {
22         ...mapGetters('users', ['users', 'getUserProductsData'])
23     },
24
25     methods: {},
26 }
27
28 </script>
29
```

The code uses Vue.js syntax like `{{ users }}` and `{{ getUserProductsData(1) }}` in the template. In the script section, it imports `mapGetters` from 'vuex'. The `computed` block uses `...mapGetters` to map getters from the 'users' module. The `users` and `getUserProductsData` getters are explicitly listed within the mapping.

Використання геттерів з модулів у компонентах

```
App.vue src M X
src > App.vue > {} "App.vue" > script
  You, 2 minutes ago | 1 author (You)
1  <template>
2  ...<div>{{ users }}</div>
3  ...<hr />
4  ...<div>{{ getUserProductsData(1) }}</div>
5  ...<hr />
6  ...<div>{{ products }}</div>
7  </template>
8
9  <script>
10 import { mapGetters } from 'vuex'
11
12 export default {
13   name: 'App',
14
15   components: {},
16
17   data() {
18     return {}
19   },
20
21   computed: {
22     ...mapGetters('users', ['users', 'getUserProductsData']),
23     ...mapGetters('products', ['products']),
24   },
25
26   methods: {},
27 }
</script>
```

```
users.js store U X
src > store > JS users.js > ...
1 const state = {
2   > users: [...],
3   ...
4 }
5
5 const getters = {
6   > users: ({ users }) => users,
7   ...
8   ...
9 }
10
11 const mutations = {
12   ...
13 }
14
15 const actions = {
16   ...
17 }
```

```
products.js store U X
src > store > JS products.js > mutations
1 const state = {
2   > products: [...],
3   ...
4 }
5
5 const getters = {
6   > products: ({ products }) => products,
7   ...
8 }
9
10 const mutations = {
11   ...
12 }
13
14 const actions = {
15   ...
16 }
```

Використання геттерів з модулів у компонентах

The diagram illustrates the flow of getters from Vuex modules to a Vue component. It consists of two main parts: a Vue component (App.vue) on the left and two Vuex modules (users.js and products.js) on the right.

App.vue (Left):

```
App.vue src M X
src > App.vue > {} "App.vue" > script
  You, 2 minutes ago | 1 author (You)
  1 <template>
  2   ...<div>{{ users }}</div>
  3   ...
  4   ...<div>{{ getUserProductsData(1) }}</div>
  5   ...
  6   ...<div>{{ products }}</div>
  7 </template>
  8
  9 <script>
 10 import { mapGetters } from 'vuex'
 11
 12 export default {
 13   name: 'App',
 14
 15   components: {},
 16
 17   data() {
 18     return {}
 19   },
 20
 21   computed: {
 22     ...mapGetters('users', ['users', 'getUserProductsData']),
 23     ...mapGetters('products', ['products']),
 24   },
 25
 26   methods: {},
 27 }
</script>
```

users.js (Right):

```
JS users.js store U X
src > store > JS users.js > ...
1 const state = {
2   > users: [...],
3   ...
4 }
5
6 const getters = {
7   > users: ({ users }) => users,
8   ...
9 }
10
11 const mutations = ...
12
13 const actions = ...
14
```

products.js (Right):

```
JS products.js store U X
src > store > JS products.js > mutations
1 const state = {
2   > products: [...],
3   ...
4 }
5
6 const getters = {
7   > products: ({ products }) => products,
8   ...
9 }
10
11 const mutations = ...
12
13 const actions = ...
14
```

The diagram uses arrows to show the flow of getters:

- A red dashed arrow points from the `users` getter in `App.vue` to the `users` getter in `users.js`.
- A green dashed arrow points from the `products` getter in `App.vue` to the `products` getter in `products.js`.

Використання **action**-методів з модулів у компонентах

Загальна схема

Приклад

```
<template>
```

```
<script>
```

```
//1. Імпортуємо mapActions
```

```
import { mapActions } from 'vuex'
```

```
export default {
```

```
}
```

```
</script>
```

```
<template>
```

```
</template>
```

```
<script>
```

```
import { mapGetters, mapActions } from 'vuex'
```

```
export default {
```

```
  name: 'App',
```

```
  computed: {
```

```
    ...mapGetters('users', ['users',  
    'getUserProductsData']),
```

```
  },
```

```
  methods: {
```

```
  },
```

```
}
```

```
</script>
```

Використання *action*-методів з модулів у компонентах

Загальна схема

Приклад

```
<template>
```

```
<script>
```

```
//1. Імпортуємо mapActions
```

```
import { mapActions } from 'vuex'
```

```
export default {
```

```
//2. Підключаємо action-методи
```

```
methods: {
```

```
...mapActions('назва_модуля', ['action-метод']),
```

```
},
```

```
.....
```

```
}
```

```
</script>
```

```
<template>
```

```
<script>
```

```
import { mapGetters, mapActions } from 'vuex'
```

```
export default {
```

```
name: 'App',
```

```
computed: {
```

```
...mapGetters('users', ['users',  
'getUserProductsData']),  
},
```

```
methods: {
```

```
...mapActions('users', ['deleteUser']),  
},
```

```
}
```

```
</script>
```

Використання *action*-методів з модулів у компонентах

Загальна схема

```
<template>
//3. Використовуємо action-методи як звичайні
methods
    . . . . .
    <тег @подія="action-метод" />
    . . . . .

</template>
```

```
<script>
//1. Імпортуємо mapActions
import { mapActions } from 'vuex'
```

```
export default {
    . . . . .
```

```
//2. Підключаємо action-методи
methods: {
    ...mapActions('назва_модуля', ['action-метод']),
},
. . . . .
```

Приклад

```
<template>
<div v-for="user in users" :key="user.id">
    <span>{{ user.name }}</span>
    <button @click="deleteUser(user.id)">Delete</button>
</div>
</template>

<script>
import { mapGetters, mapActions } from 'vuex'

export default {
    name: 'App',
    computed: {
        ...mapGetters('users', ['users',
            'getUserProductsData']),
    },
    methods: {
        ...mapActions('users', ['deleteUser']),
    },
}</script>
```

Використання action-методів з модулів у компонентах

The screenshot shows two code editors side-by-side, illustrating the flow of actions from a Vue component to a Vuex store.

Left Editor (App.vue):

```
src > App.vue > {} "App.vue" > template > div
  You, 1 second ago | 1 author (You)
  1 <template>
  2   <div v-for="user in users" :key="user.id">
  3     <span>{{ user.name }}</span>
  4     <button @click="deleteUser(user.id)">Delete</button>
  5   </div>
  6 </template>
  7
  8 <script>
  9   import { mapGetters, mapActions } from 'vuex'
 10
 11   export default {
 12     name: 'App',
 13
 14     computed: {
 15       ...mapGetters('users', ['users', 'getUserProductsData']),
 16     },
 17
 18     methods: {
 19       ...mapActions('users', ['deleteUser']),
 20     },
 21   }
 22 </script>
```

A red dashed arrow points from the `deleteUser` method in the `methods` block of `App.vue` to the `deleteUser` action in the `mutations` block of the `users.js` store module.

Right Editor (users.js store):

```
src > store > users.js > mutations
  ...
  1 const state = {
  2   users: [...],
  3 }
  4
  5 const getters = {
  6   users: (state) => state.users,
  7   getUserProductsData: (state, getters, rootState, rootGetters) => (userId) => {
  8     ...
  9   },
 10 }
 11
 12 const mutations = {
 13   ...
 14 }
 15
 16 const actions = {
 17   addUser(context, user) {
 18     ...
 19   },
 20   editUser(context, editedUser) {
 21     ...
 22   },
 23   deleteUser(context, userId) {
 24     ...
 25   },
 26 }
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50 }
```

A red arrow points from the `deleteUser` action in the `mutations` block to the `deleteUser` action in the `actions` block.

Як звернутись до іншого геттера в межах одного локального модуля

```
import { createStore } from 'vuex'

const store = createStore({
    . . . . .
    //опис геттерів
    getters: {
        інший_геттер (state) {
            . . . . .
            //звертання до іншого геттера
            getters
            . . . . .
        },
        геттер1(state, getters) {
            . . . . .
            //звертання до іншого геттера
            getters . інший_геттер
            . . . . .
        },
        . . . . .
    },
    . . . . .
})
```

Використання геттерів з інших модулів

```
import { createStore } from 'vuex'

const store = createStore({
    . . . . .
    //опис обчисл.власт.(аналог computed)
    getters: {
        обч.геттер1(state, getters, rootState, rootGetters) {
            . . . .
            rootGetters('назва_модуля / геттер ')
        },
        . . . .
    },
    . . . .
})
```



```
JS users.js store U X
src > store > JS users.js > [0] getters
1 const state = {
2     users: [
3         { id: 1, name: 'Користувач 1', productsList: [1, 3] },
4         { id: 2, name: 'Користувач 2', productsList: [2, 4] },
5         { id: 3, name: 'Користувач 3', productsList: [1, 5] },
6         { id: 4, name: 'Користувач 4', productsList: [2, 3, 4] },
7         { id: 5, name: 'Користувач 5', productsList: [5] },
8     ],
9 }
10
11 const getters = {
12     users: ({ users }) => users,
13     getUserProductsData: (state, getters, rootState, rootGetters) => (userId) => {
14         const user = state.users.find(u => u.id === userId)
15         if (user) {
16             const userProducts = rootGetters['products/products'].filter(p => user.productsList.includes(p.id))
17             return userProducts
18         }
19         return []
20     },
21 }
22
23 > const mutations = ...
24
25 > const actions = ...
26 }
```

JS users.js store U X

src > store > JS users.js > [o] getters

```
1  const state = {  
2   ...users: [  
3     { id: 1, name: 'Користувач 1', productsList: [1, 3] },  
4     { id: 2, name: 'Користувач 2', productsList: [2, 4] },  
5     { id: 3, name: 'Користувач 3', productsList: [1, 5] },  
6     { id: 4, name: 'Користувач 4', productsList: [2, 3, 4] },  
7     { id: 5, name: 'Користувач 5', productsList: [5] },  
8   ],  
9 }  
10  
11 const getters = {  
12   ...users: ({ users }) => users,  
13   getUserProductsData: (state, getters, rootState, rootGetters) => (userId) => {  
14     const user = state.users.find((u) => u.id === userId)  
15     if (user) {  
16       const userProducts = rootGetters['products/products'].  
17         filter((p) => user.productsList.includes(p.id))  
18       return userProducts  
19     }  
20   },  
21 }  
22  
23 > const mutations = { ...  
24  
25 > const actions = { ...  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40 }
```

⌚ ⚡

JS products.js store U X

src > store > JS products.js > [o] getters > [o] products

```
1  const state = {  
2   ...products: [  
3     { id: 1, title: 'Продукт 1', price: 100 },  
4     { id: 2, title: 'Продукт 2', price: 200 },  
5     { id: 3, title: 'Продукт 3', price: 150 },  
6     { id: 4, title: 'Продукт 4', price: 120 },  
7     { id: 5, title: 'Продукт 5', price: 180 },  
8   ],  
9 }  
10  
11 const getters = {  
12   ...products: ({ products }) => products,  
13 }  
14 > const mutations = { ...  
15 }  
16  
17 > const actions = { ...  
18 }  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40 }
```



```
import { createStore } from 'vuex'

const store = createStore({
    . . . . .
    //опис методів
    actions: {
        інший_action_метод (... ) {
            . . . . .
        },
        action_метод({ dispatch, commit }) {
            dispatch(
                'інший_action_метод',
                параметри
            )
        }
    . . . . .
    },
    . . . . .
})

```



Використання action-методів з інших модулів

```
import { createStore } from 'vuex'

const store = createStore({
    . . . . .
    //опис методів
    actions: {
        action_метод({ dispatch, commit, getters, rootGetters }) {
            dispatch(
                'назва_модуля / назва_action_метода',
                параметри,
                { root: true }
            )
        }
    . . . . .
},
. . . . .
})
```

```
const state = {
    products: [ ... ],
}
const getters = {
}
const mutations = {
}

const actions = {
    editProduct(context, editedProduct) {
        . . .
    },
    deleteProduct({ commit, dispatch }, productId) {
        commit('deleteProduct', productId)
        dispatch(
            'users/deleteProductFromUsers',
            productId,
            { root: true }
        )
    }
}
```

Використання action-методів з інших модулів

JS products.js store U ●

src > store > JS products.js > [o] actions > ⚙ deleteProduct

```
1  const state = {  
2    >   products: [...]  
8    ...  
9  }  
10 > const getters = { ...  
12  }  
13 > const mutations = { ...  
23  }  
24  
25  const actions = {  
26    >   editProduct(context, editedProduct) { ...  
28    },  
29    >   deleteProduct({ commit, dispatch }, productId) {  
30      commit('deleteProduct', productId)  
31      dispatch(  
32        >   'users/deleteProductFromUsers',  
33        >   productId, {  
34          >   root: true  
35        })  
36    },  
37  }  
38
```

JS users.js store U X

src > store > JS users.js > [o] default

```
1  const state = {  
2    >   users: [...]  
8    ...  
9  }  
10  
11 > const getters = { ...  
21  }  
22  
23 > const mutations = { ...  
36  }  
37  
38  const actions = {  
39    >   addUser(context, user) { ...  
43    },  
44    >   editUser(context, editedUser) { ...  
46    },  
47    >   deleteUser(context, userId) { ...  
49    },  
50    >   deleteProductFromUsers({ commit }, productId){  
51      >   commit('removeProductFromUsers', productId)  
52    },  
53  }
```

JS index.js store M X

```
src > store > JS index.js > [o] store > ⚡ mutations
  You, 15 minutes ago | 1 author (You)
  import { createStore } from 'vuex'
  import productsModule from './products'
  import usersModule from './users'

  // Create a new store instance.
  const store = createStore({
    state: () => {
      return {
        globalCounter: 99,
      }
    },
    getters: {
      getGlobalCounter: (state) =>
        state.globalCounter,
    },
    mutations: {
      setGlobalCounter(state) {
        state.getGlobalCounter++
      },
    },
    actions: {
      incGlobalCounter({ commit }) {
        commit('setGlobalCounter')
      },
    },
    modules: {
      products: productsModule,
      users: usersModule,
    },
  })
```

JS products.js store U X

```
src > store > JS products.js > [o] mutations > 📄 editProduct > [e]
  1  const state = {
  2    > ... products: [...],
  3    ... ],
  4    ...
  5  }
  6  const getters = {
  7    ... products: ({ products }) => products,
  8    ...
  9  }
```

rootState

```
▼ Proxy(Object) {globalCounter: 99, products
  ► [[Handler]]: MutableReactiveHandler
  ▼ [[Target]]: Object
    globalCounter: 99
    ▼ products:
      ► products: (4) [Proxy(Object), Proxy(Object), Proxy(Object), Proxy(Object)]
      ► [[Prototype]]: Object
    ▼ users:
      ► users: (5) [{...}, {...}, {...}, {...}, {...}]
      ► [[Prototype]]: Object
```

JS users.js store U X

```
src > store > JS users.js > [o] getters
  1  const state = {
  2    > ... users: [...],
  3    ... ],
  4    ...
  5  }
  6  const getters = {
  7    ... getUsers: ({ users }) => users,
  8    ...
  9  }
  10
  11  const mutations = {
  12    ...
  13  }
  14
  15  > const actions = {
  16    ... addUser(context, user) {
  17      ...
  18    },
  19    ... editUser(context, editedUser) {
  20      ...
  21    },
  22    ... deleteUser(context, userId) {
  23      ...
  24    },
  25    ... deleteProductFromUsers(context, productId) {
  26      ...
  27    },
  28  }
```

JS index.js store M X

```
src > store > JS index.js > [0] store > ⚡ mutations
  You, 15 minutes ago | 1 author (You)
  1 import { createStore } from 'vuex'
  2 import productsModule from './products'
  3 import usersModule from './users'
  4
  // Create a new store instance.
  5 const store = createStore({
  6   state: () => {
  7     return {
  8       globalCounter: 99,
  9     }
 10   },
 11   getters: {
 12     getGlobalCounter: (state) => state.globalCounter,
 13   },
 14   mutations: {
 15     setGlobalCounter(state) {
 16       state.getGlobalCounter++
 17     },
 18   },
 19   actions: {
 20     incGlobalCounter({ commit }) {
 21       commit('setGlobalCounter')
 22     },
 23   },
 24   modules: {
 25     products: productsModule,
 26     users: usersModule,
 27   },
 28 },
 29 )
```

JS products.js store U X

```
src > store > JS products.js > [0] mutations > ⚡ editProduct > [e]
  1 const state = {
  2   products: [...],
  3 }
  4 const getters = {
  5   products: ({ products }) => products,
  6 }
  7 const mutations = {
  8   editProduct(state, editedProduct) {
  9     const index = state.products.findIndex(p => p.id === editedProduct.id)
 10     if (index !== -1) {
 11       state.products[index] = editedProduct
 12     }
 13   },
 14   deleteProduct(state, productId) {
 15     state.products = state.products.filter(p => p.id !== productId)
 16   },
 17 }
 18 const actions = {
 19   editProduct(context, editedProduct) {
 20     context.commit('editProduct', editedProduct)
 21   },
 22   deleteProduct({ commit, dispatch, rootState, rootGetters }, productId) {
 23     dispatch('editProduct', { ...rootState.products.find(p => p.id === productId), id: null })
 24     commit('deleteProduct', productId)
 25   },
 26 }
 27 
```

rootGetters

▼ {} ⓘ

getGlobalCounter: (...)

products/products: (...)

users/getUserProductsData: (...)

users/getUsers: (...)

users/users: (...)

JS users.js store U X

```
src > store > JS users.js > [0] getters
  1 const state = {
  2   users: [...],
  3 }
  4 const getters = {
  5   getUsers: ({ users }) => users,
  6   users: ({ users }) => users,
  7   getUserProductsData: (state, getters, rootState, rootGetters) => (userId) => {
  8     const user = getters.users.find(u => u.id === userId)
  9     if (!user) return []
 10     const products = user.products.map(productId => {
 11       const product = rootState.products.find(p => p.id === productId)
 12       if (!product) return {}
 13       return { ...product, user }
 14     })
 15     return products
 16   },
 17 }
 18 const mutations = {
 19   addUser(context, user) {
 20     context.commit('addUser', user)
 21   },
 22   editUser(context, editedUser) {
 23     context.commit('editUser', editedUser)
 24   },
 25   deleteUser(context, userId) {
 26     context.commit('deleteUser', userId)
 27   },
 28   deleteProductFromUsers(context, productId) {
 29     context.commit('deleteProductFromUsers', productId)
 30   },
 31 }
 32 const actions = {
 33   addNewUser({ commit, dispatch, rootState, rootGetters }, user) {
 34     const { name, email, password } = user
 35     const hashedPassword = rootGetters['hasher'](password)
 36     const newUser = { name, email, password: hashedPassword }
 37     commit('addUser', newUser)
 38     dispatch('editUser', { ...newUser, id: newUser.id })
 39   },
 40   editUser({ commit, dispatch, rootState, rootGetters }, user) {
 41     const { id, name, email, password } = user
 42     const hashedPassword = rootGetters['hasher'](password)
 43     const editedUser = { id, name, email, password: hashedPassword }
 44     commit('editUser', editedUser)
 45   },
 46   deleteUser({ commit, dispatch, rootState, rootGetters }, userId) {
 47     const user = rootState.users.find(u => u.id === userId)
 48     if (!user) return
 49     commit('deleteUser', userId)
 50     dispatch('deleteProductFromUsers', user.products)
 51   },
 52   deleteProductFromUsers({ commit, dispatch, rootState, rootGetters }, productId) {
 53     const products = rootState.products.filter(p => p.id !== productId)
 54     commit('products', products)
 55     dispatch('editUser', { ...rootState.users.find(u => u.products.includes(productId)), products })
 56   },
 57 }
```