Vue Fundamentals – 75
Working with Vuex Modules

Peter Kassenaar –
info@kassenaar.com

Store gets B.I.G.

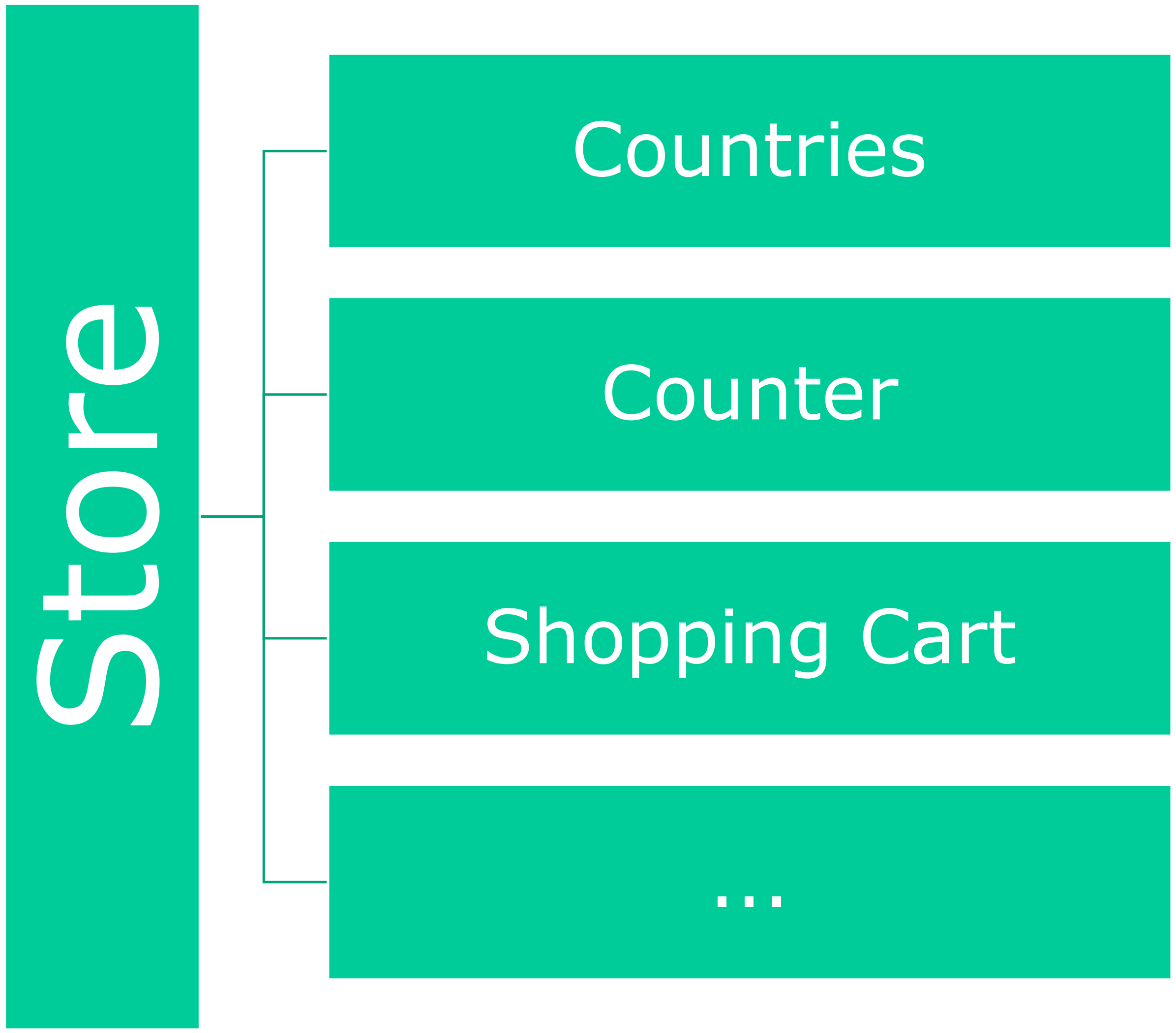
"Due to using a single state tree, all state of our application is contained inside one big object. However, as our application grows in scale, the store can get really bloated."

https://vuex.vuejs.org/guide/modules.html

Solution – divide store in Modules

- Each Module:
  - Has its own state
  - Has its own mutations
  - Has its own actions
  - Has its own getters
  - Can even have sub/child modules
- Often: also split up the directory tree

Store modules

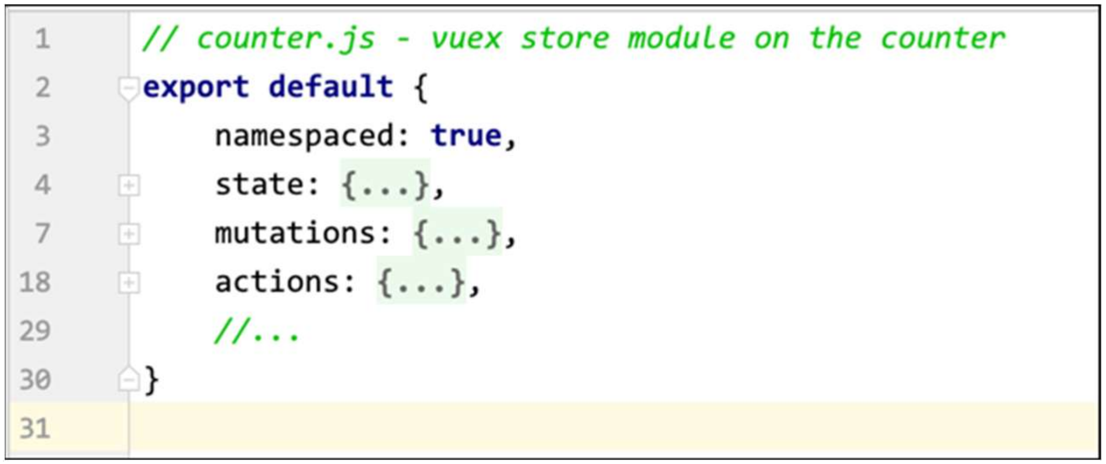


1. Add `<module>.js`-files for every module

Create `.js`-files for every module in the store.

Design the modules as always:

New: notice the use of the `namespaced` attribute

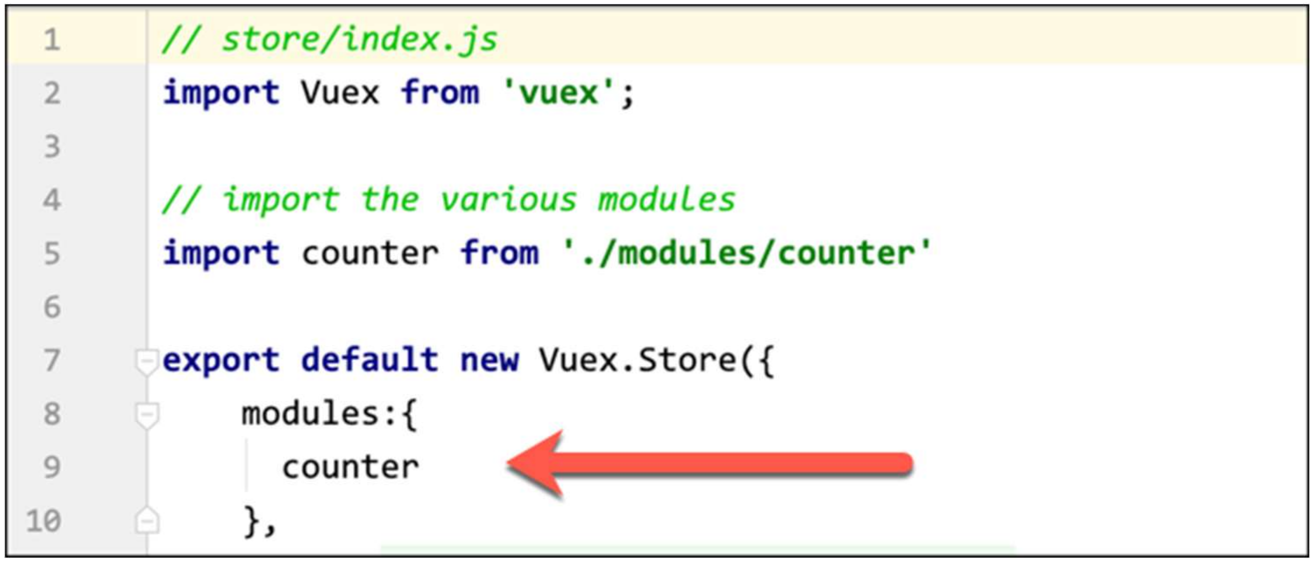


```
// counter.js - vuex store module on the counter
export default {
    namespaced: true,
    state: {...},
    mutations: {...},
    actions: {...},
    //...
}

```

2. Add the module to the store

- Add a modules object to the store, pointing to the imported modules
- Notice the shorthand notation
  - It's the same as `modules: { counter : counter }`



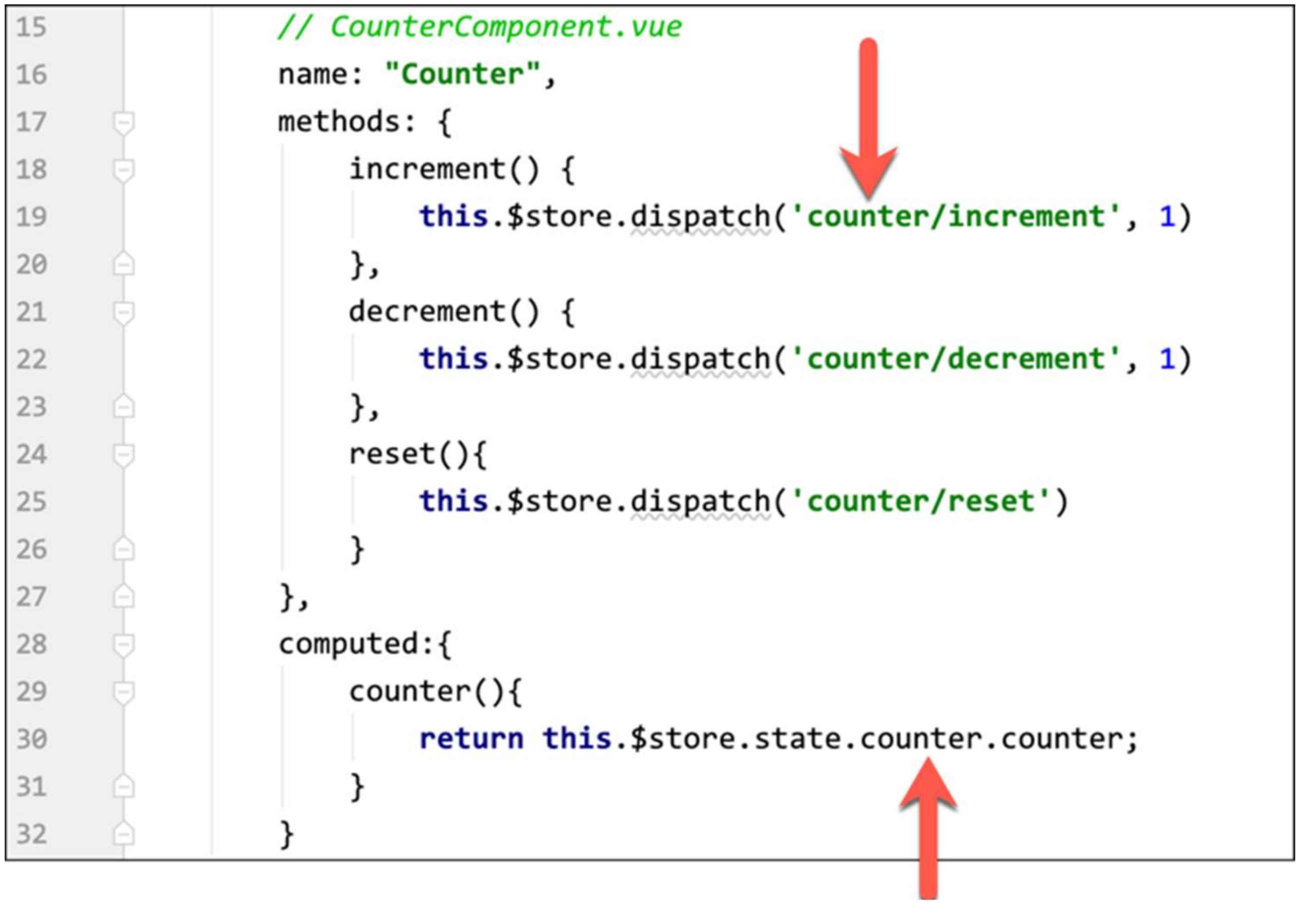
```
// store/index.js
import Vuex from 'vuex';

// import the various modules
import counter from './modules/counter'

export default new Vuex.Store({
    modules:{
      counter
    },
```

3. Update the components

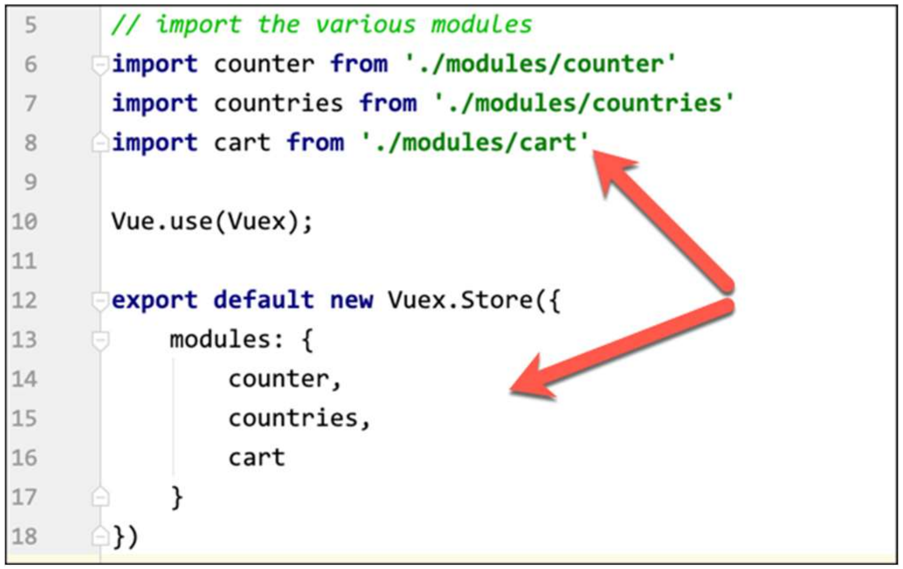
- Add/use the module name as the namespace for your store actions



```
// CounterComponent.vue
name: "Counter",
methods: {
    increment() {
        this.$store.dispatch('counter/increment', 1)
    },
    decrement() {
        this.$store.dispatch('counter/decrement', 1)
    },
    reset(){
        this.$store.dispatch('counter/reset')
    }
},
computed:{
    counter(){
        return this.$store.state.counter.counter;
    }
}
```

4. Create more modules

- Move/create every part of the store to its own module.
- In our example we have three modules:
  - `Countries`
  - `Cart`
  - `Counter`
- The store becomes rather simple:



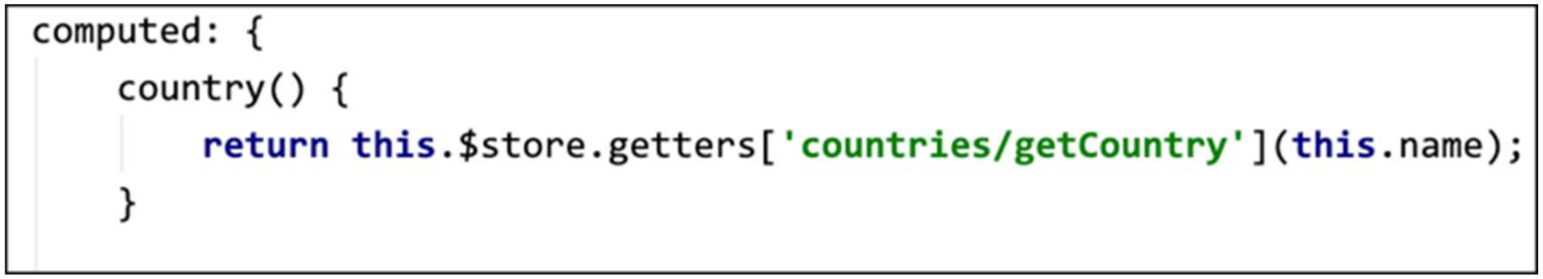
```
// import the various modules
import counter from './modules/counter'
import countries from './modules/countries'
import cart from './modules/cart'

Vue.use(Vuex);

export default new Vuex.Store({
    modules: {
        counter,
        countries,
        cart
    }
})
```

Accessing getters

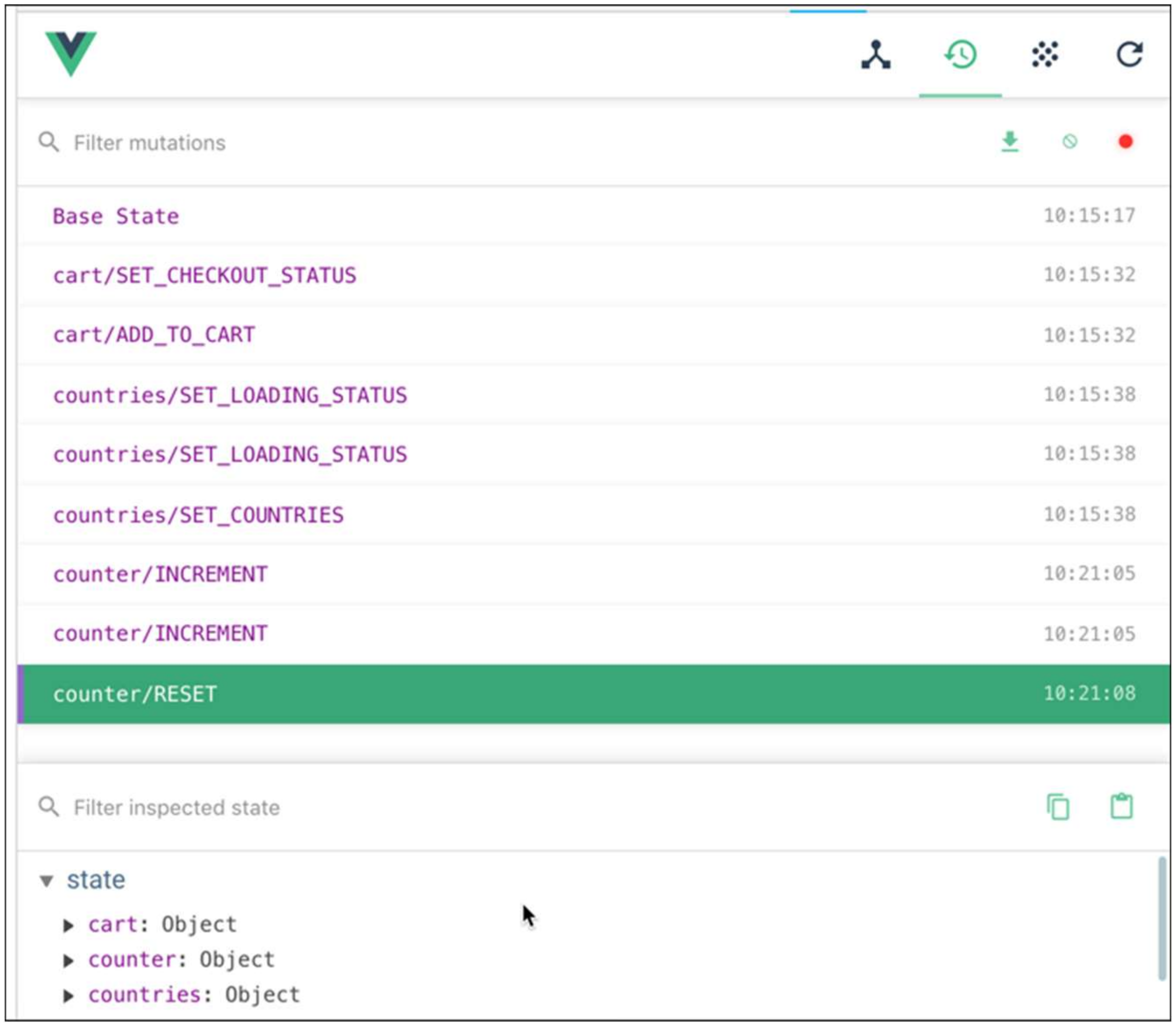
- Accessing getters from the store is a bit more cumbersome
- Notice the (ugly!) notation of passing parameters to the getter
- For instance, in the `countries.js` module:



```
computed: {
    country() {
        return this.$store.getters['countries/getCountry'](this.name);
    }
```

Using the Vue DevTools

- Store modules are also visible in the Vue DevTools:

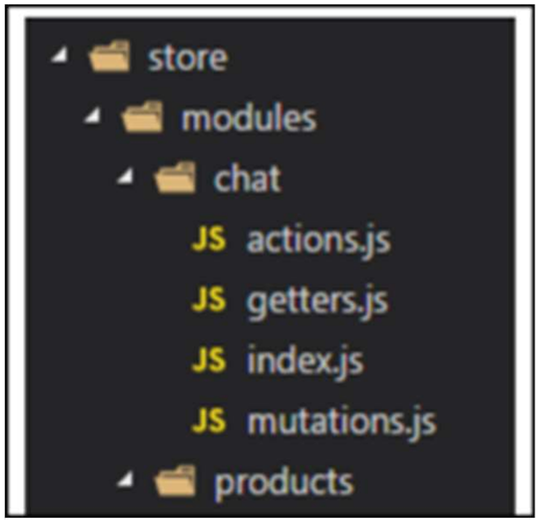


```
Base State                          10:15:17
cart/SET_CHECKOUT_STATUS            10:15:32
cart/ADD_TO_CART                    10:15:32
countries/SET_LOADING_STATUS        10:15:38
countries/SET_LOADING_STATUS        10:15:38
countries/SET_COUNTRIES             10:15:38
counter/INCREMENT                   10:21:05
counter/INCREMENT                   10:21:05
counter/RESET                       10:21:08
```

```
▼ state
  ▶ cart: Object
  ▶ counter: Object
  ▶ countries: Object
```

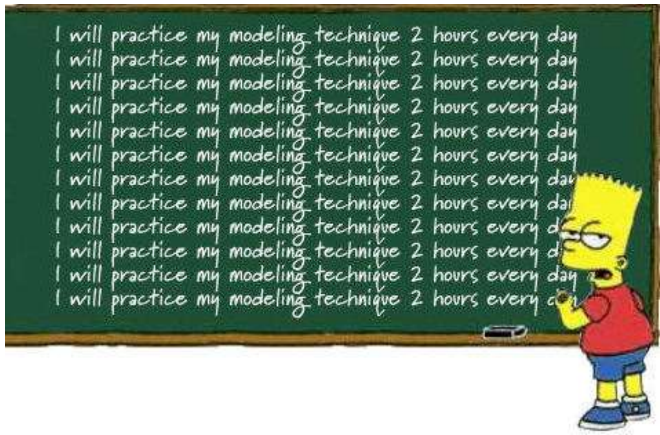
Best Practices

- Use `vuex` if your application has more than 3-4 components, sharing data.
- Use vuex store modules from the start.
- Large module? Split up `actions.js`, `mutations.js`, etc.
- Every feature should have its own store module

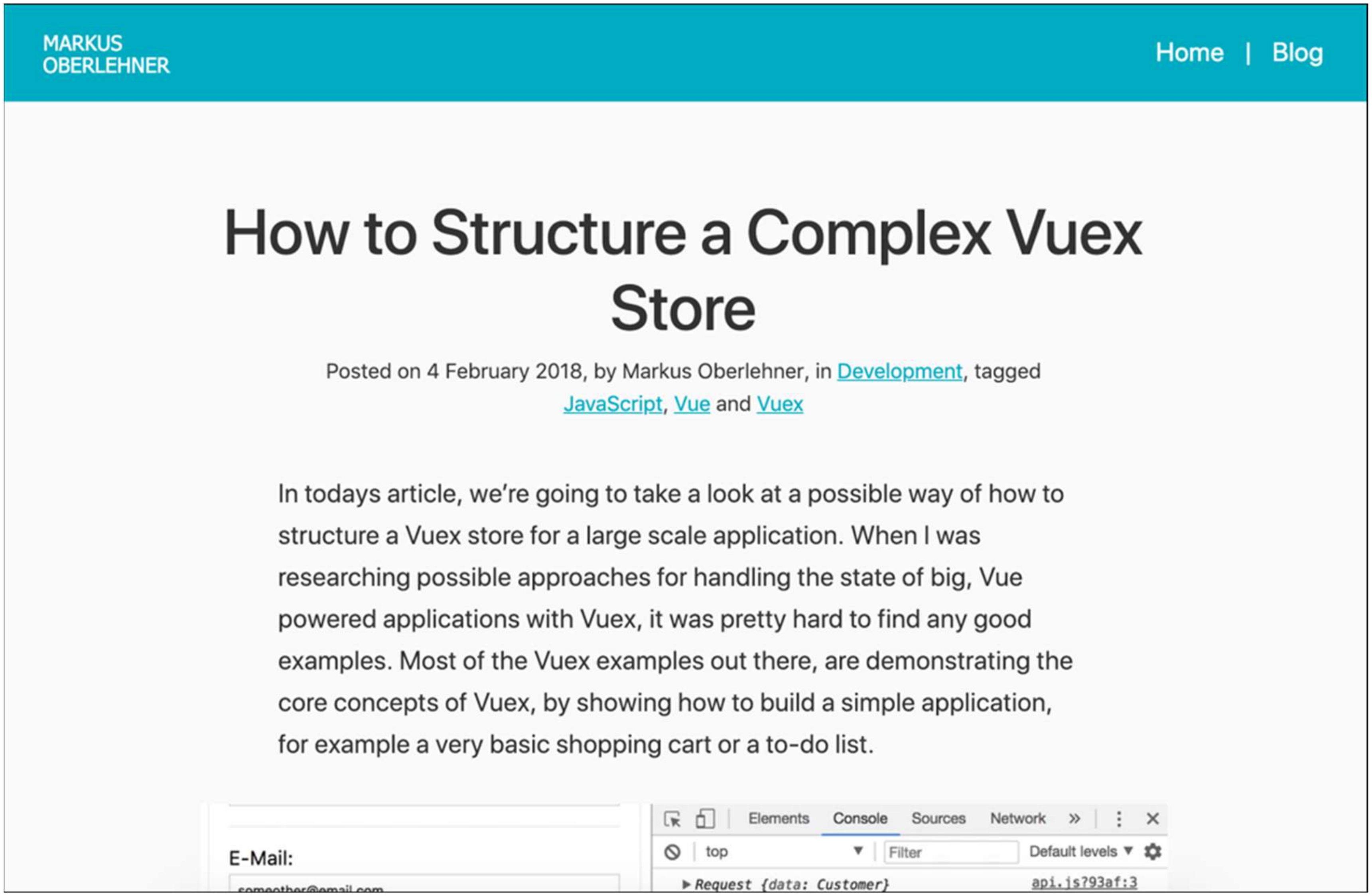


Workshop

- Update the store in your own application to use modules
- Follow the steps in this presentation to add modules
  - Move logic from central store to feature modules
- In development – you can have a store partial filled with modules, and partial filled with 'direct' data.
  - No need for a 'big bang' – you can upgrade gradually
- Optional – set a (new) property in the store, `selectedCountry` if you click a specific country
- Generic example `../430-vuex-modules`



More info


MARKUS OBERLEHNER

Home | Blog

How to Structure a Complex Vuex Store

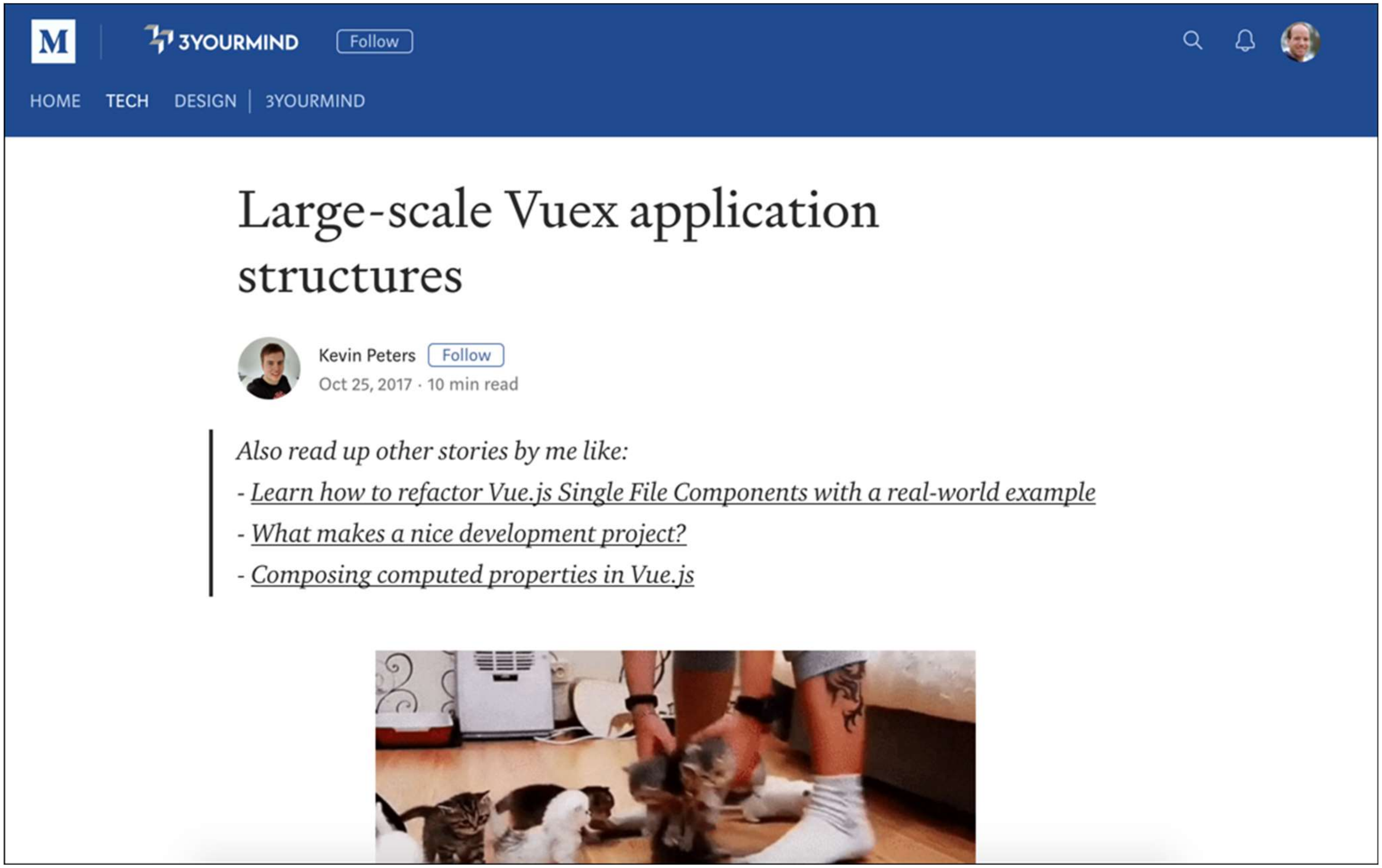
Posted on 4 February 2018, by Markus Oberlehner, in Development, tagged JavaScript, Vue and Vuex

In todays article, we're going to take a look at a possible way of how to structure a Vuex store for a large scale application. When I was researching possible approaches for handling the state of big, Vue powered applications with Vuex, it was pretty hard to find any good examples. Most of the Vuex examples out there, are demonstrating the core concepts of Vuex, by showing how to build a simple application, for example a very basic shopping cart or a to-do list.

E-Mail:

```
▶ Request {data: Customer}                api.js?93af:3
```

https://markus.oberlehner.net/blog/how-to-structure-a-complex-vuex-store/



Large-scale Vuex application structures

Kevin Peters Follow
Oct 25, 2017 · 10 min read

> *Also read up other stories by me like:*
> - *Learn how to refactor Vue.js Single File Components with a real-world example*
> - *What makes a nice development project?*
> - *Composing computed properties in Vue.js*

https://medium.com/3yourmind/large-scale-vuex-application-structures-651e44863e2f

Checkpoint

- You know the benefits of using store modules
- You know how to split up your store into modules
- You know how to import and use modules
- You know about accessing store modules and namespaces