# Api

Each endpoint is a model? So like getEmpoyeeById(id). How do you do the typing of the “employee” object then? (and where does the typings live inside the employee store? – where the getEmpoyeeById is?)

In my demo case clicking on a tab does a call to the api.

I wanted to build a “wrapper” around the api call to do a few things:

1. It needs to check if another call is currently open to the api, if another call (via name if provided – “get-empoyee-request” – or if not provided then to md5(url + params). If it exists in the “requests” store then to cancel the previous 1. To stop something like <https://puu.sh/Ib5To/9da6ecab03.gif>
2. on every response from the api I attach a “profiler” key. (if in debug mode). The api “wrapper” then dispatches a profiler/add call with the profiler info that then gets stored in the profiler store. <https://puu.sh/Ib5WM/5f803502e0.png> (the api is purposely sleeping for a second hence the response times)
3. the response then deletes the PROFILER key from the response and hands it to the variable to be used wherever without a huge debug dump)

point 2 and 3 are implemented already. I have the option of the name and key for point 1 but I haven’t implemented the cancel() part yet.

I don’t know what its supposed to be called (the wrapper) thing. So I stuck it into composable. Its definitely not a mixin. Might be a plugin. Don’t think it’s a component. Its something that gets used by other things. Like a helper function?

# State

On making any change on the screen like clicking the tab button to set the query string to that actions value. (not move to a separate page for instance). Think ordering of a table. Click the sort button and it sets ?order=name on the query string. Doing /api/employees/list/:order (name, email, age, roles etc) seems like a bad idea. (what if theres a search? Order, filter by role then the url would get interesting, easier just going if you change the data this.search then set the address bar to ?search=xyz&order=name&role=admin etc. then pass all those to the api to as params.

If I purely stick to the reactivity part (data() ) then I loose the “click refresh in the browser” or open in new tab stuff.

Was thinking of something along the lines of state.set(“key”,”value”) and that sets the query string to ?key=value and returns the value so that I can do this.something = state.set(“key”,”value”) and it now holds value. On changing “key” in the address abr it changes the this.something.

I was also thinking that on changing “key” at this point to emit an event of like to call a function to “go fetch the data again from the api again but use these options this time”.

In my demo thing I have 4 tabs. Each tab needs to go to the api and return data for it. would you set it up as 4 store getters? getTabHome() getTabLogs() etc? or a single getTab(“home”) type thing? (each tab will return different “data” keys and values)

Using the vuex store to call the api takes would solve my early issue, “how to access the store state inside an “external” (state.set()) function/plugin since it doesn’t have access to this.$store. It would also be a good place for typings since its “I will return x” no matter if its from my internal or I first got to fetch from api. It just seemed “off” for me, store should be store, action should be action etc. (not confusing the “action” on the store like mutation / action etc). think I’ll give this a try tho. (now im confused again. Thought the api call was in /api/ but I thought it was in /store/)

So whats the flow again?

/api/ has the “sections” you might be interested in, ie “front” / “admin” (or finer grained). For front have a /api/front/home.ts file with “getWelcomeMessage()” and “getRecomendedPosts()” methods (that call the api)

store then uses const welcome\_message = frontHome.getWelcomeMessage() and commits its state. Is this a getter or action?

The .vue file then uses the store data for display (this.$store.getters.home() ?

Im pretty sure you just gonna recommend something like

data() {

search: undefined,  
 order:undefined,  
 role: “all”,

list: []  
}  
methods: {  
 async getList(){  
 this.list = await this.$store.getEmployeeByName(search,order,role)  
 }  
}  
// don’t know how this works yet  
watch() this.search, this.order, this.role {  
 this.getList()  
  
}

but that doesn’t account for “refresh” on page. or new tab. For now I can probably live without those. But its definitely something I wanted in tho.