(https://www.nightprogrammer.com/)

Ask

7 Comments (https://www.nightprogrammer.com/vue-js/architect-structure-large-scale-vue-js-application-example/#comments)
Featured (https://www.nightprogrammer.com/category/featured/) / Vue.js (https://www.nightprogrammer.com/category/vue-js/)
GautamK (https://www.nightprogrammer.com/author/gautam/)
November 14, 2021

Wondering what is the best way to structure your Vue.js application so that it remains maintainable and extendable? The answer to that lies in modularity and predictability. You'd think that Vue is all about module based components and state management. However, it is much more than that. It is very powerful by performance (beats React by benchmarks). And the way you structure your project can make a lot of difference. You'll find numerous articles and examples over the web on this. However, most of them will only show a flat styled directory structure without explaining how they communicate with each other. Or how you'll need to scaffold it as your project grows rapidly.

I've tried and tested multiple structures over time, with different (large scale) projects. And have come to the conclusion that every project demands a different kind of structure depending on the project requirements. The degree of differentiation, of course, varies. There's no ideal project structure that can fit in any project you throw in it. The diversity in project structure increases with the project size. Which means, you'll eventually need to make slight changes to your project structure as it grows. However, if you understand how to architect your project from the get-go so that you won't need to make breaking changes, your project structure will remain maintainable and extendible.

Atomic design

There are multiple design patterns available across the web. There's one that stands out to me. The atomic design (https://bradfrost.com/blog/post/atomic-web-design/) pattern by Brad Frost. Although, the main idea behind the system article. I will, however, be showing an architectural implementation using the application part of the guideline.

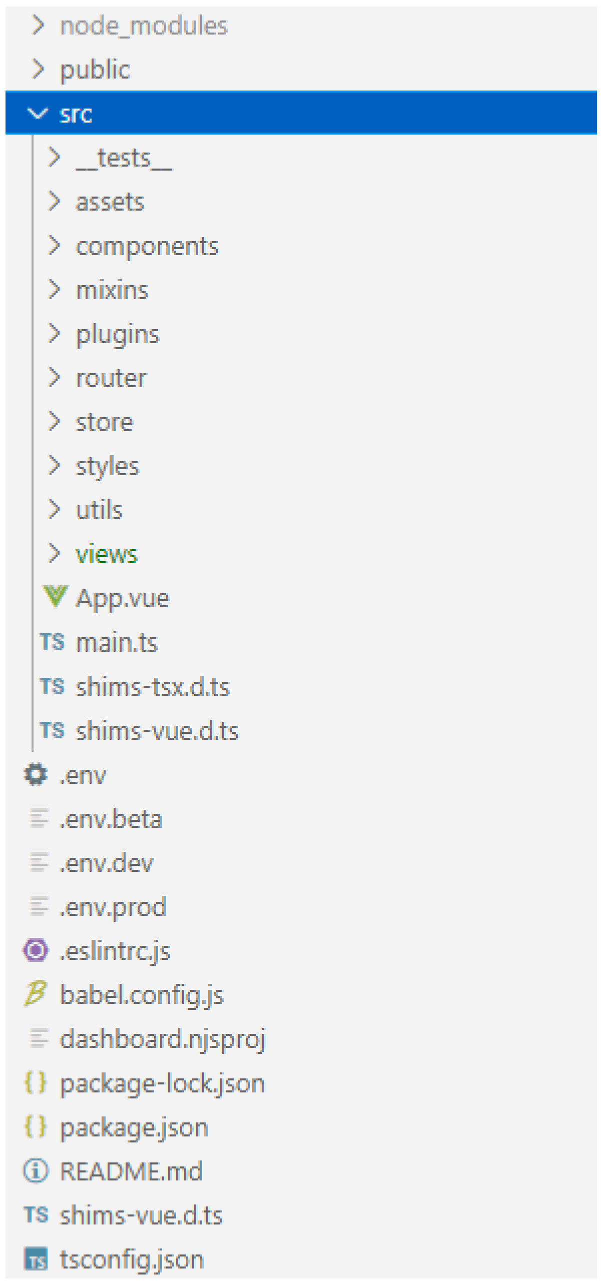
(https://www.nightprogrammer.com/)

Other than the design system, a framework definitely help you kickstart your project with tons of ready-to-use components out of the box. There are multiple popular Vue.js frameworks, like Vuetify, Quasar or Buefy. The architecture I'm going to explain here is applicable to all the Vue frameworks if you choose to use one.

Ask

Here I'm showing you the most condensed form of the structure. We'll go through each of the module and see how they extend further.

Project structure example



```
> node_modules
> public
v src
  > __tests__
  > assets
  > components
  > mixins
  > plugins
  > router
  > store
  > styles
  > utils
  > views
  App.vue
  main.ts
  shims-tsx.d.ts
  shims-vue.d.ts
.env
.env.beta
.env.dev
.env.prod
.eslintrc.js
babel.config.js
dashboard.njsproj
package-lock.json
package.json
README.md
shims-vue.d.ts
tsconfig.json
```

some developers prefer to keep the tests directory at the top of hierarchy. You can use your favourite testing library (Jest, Mocha, etc). Personally I use vue-test-utils (official testing library provided by Vue.js).

Assets

You can keep all your static assets like images, SVGs, sprites here. However, if there's an SVG in which you can pass a text or a numeric value through props, it is not a static anymore. You want to keep those assets inside the components directory instead.

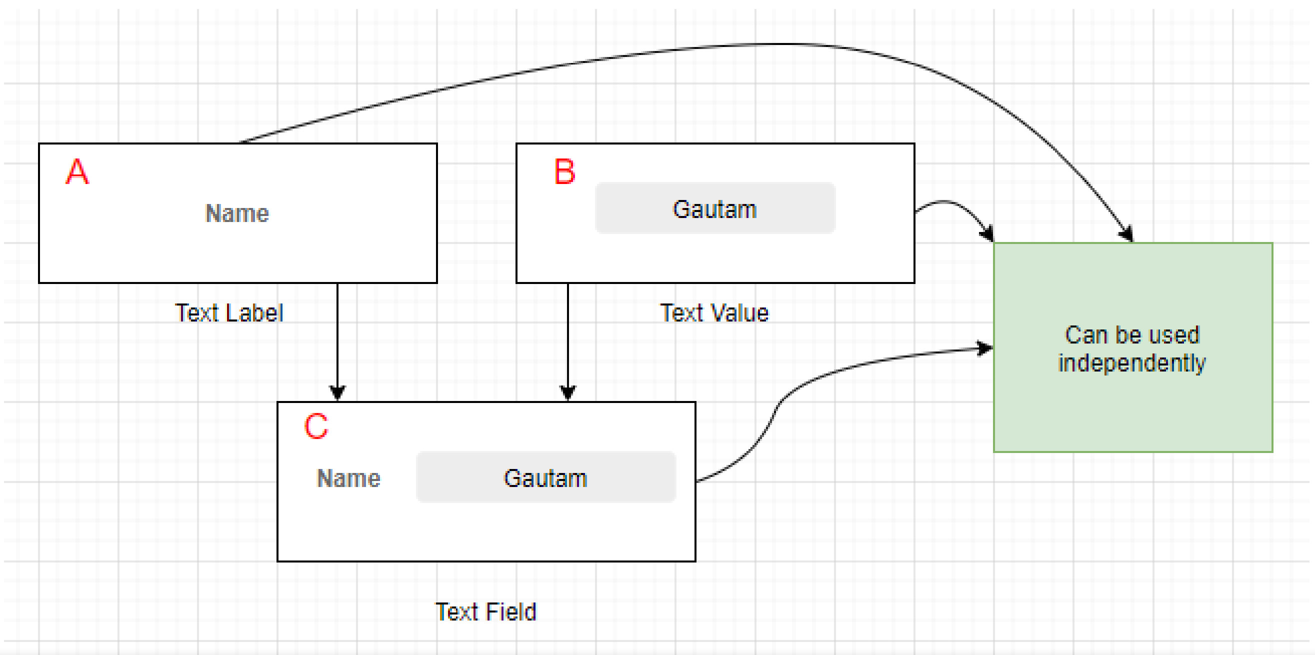
Components

Ask

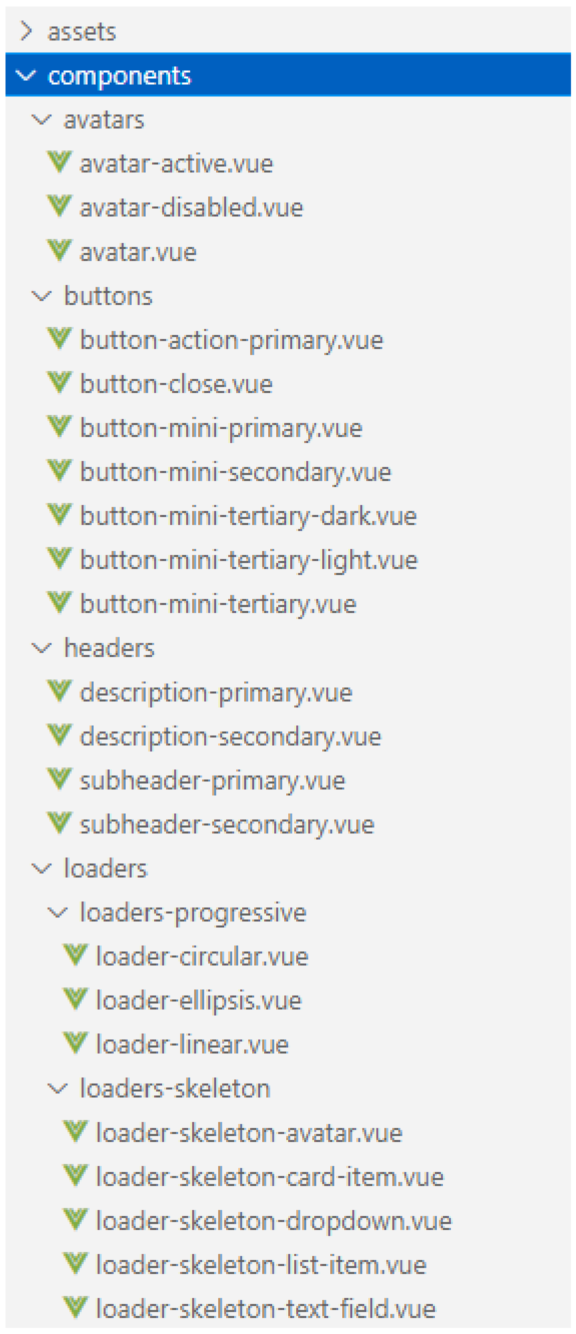
This is probably one of the most important module of your project. As it provides the building blocks of your application. This is also a module where you can apply the atomic design idea. Let me tell you what that is in brief, here:

Keep in mind that rules of chemistry do not exactly apply here, but some of it do. Consider there is an atom A, and there is an atom B. We can use the atom A wherever we want, same with atom B. We can also combine atom A and atom B and create a new molecule C. It is important to understand here that molecule C will now have the properties of both atom A and B but, A and B can still function on their own as earlier. We can now use this C molecule wherever we want. Again, this C molecule can further combine with other atoms or molecules to give other derivatives.

Here, I've created a diagram to help you visualize the idea:



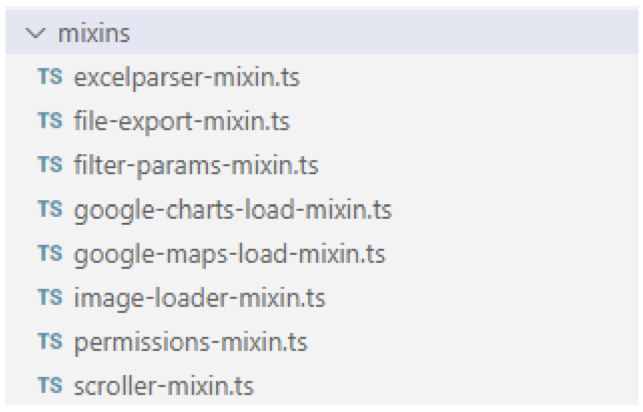
independently as well. This is only one example of component based atomic design system. This is applicable to a large part of your project. Now, let me show you how you can structure your component directory:



```
> assets
v components
  v avatars
      avatar-active.vue
      avatar-disabled.vue
      avatar.vue
  v buttons
      button-action-primary.vue
      button-close.vue
      button-mini-primary.vue
      button-mini-secondary.vue
      button-mini-tertiary-dark.vue
      button-mini-tertiary-light.vue
      button-mini-tertiary.vue
  v headers
      description-primary.vue
      description-secondary.vue
      subheader-primary.vue
      subheader-secondary.vue
  v loaders
    v loaders-progressive
        loader-circular.vue
        loader-ellipsis.vue
        loader-linear.vue
    v loaders-skeleton
        loader-skeleton-avatar.vue
        loader-skeleton-card-item.vue
        loader-skeleton-dropdown.vue
        loader-skeleton-list-item.vue
        loader-skeleton-text-field.vue
```

This screenshot is only a part of the components I've created, as it is a long list. You'll notice that I've created many of the components using the atomic design idea. Whenever I find that there are multiple components sharing more than one properties (props), I categorize them into sub-folders (maintaining modularity). You can see the same in case of loaders-progressive and loaders-skeleton folders above.

Mixins



```
v mixins
    TS excelparser-mixin.ts
    TS file-export-mixin.ts
    TS filter-params-mixin.ts
    TS google-charts-load-mixin.ts
    TS google-maps-load-mixin.ts
    TS image-loader-mixin.ts
    TS permissions-mixin.ts
    TS scroller-mixin.ts
```

Mixins can help you save a lot of repetitive coding. Whenever you find that there are operations that are being repeated at multiple places, create a mixin

Ask

(https://www.nightprogrammer.com/)

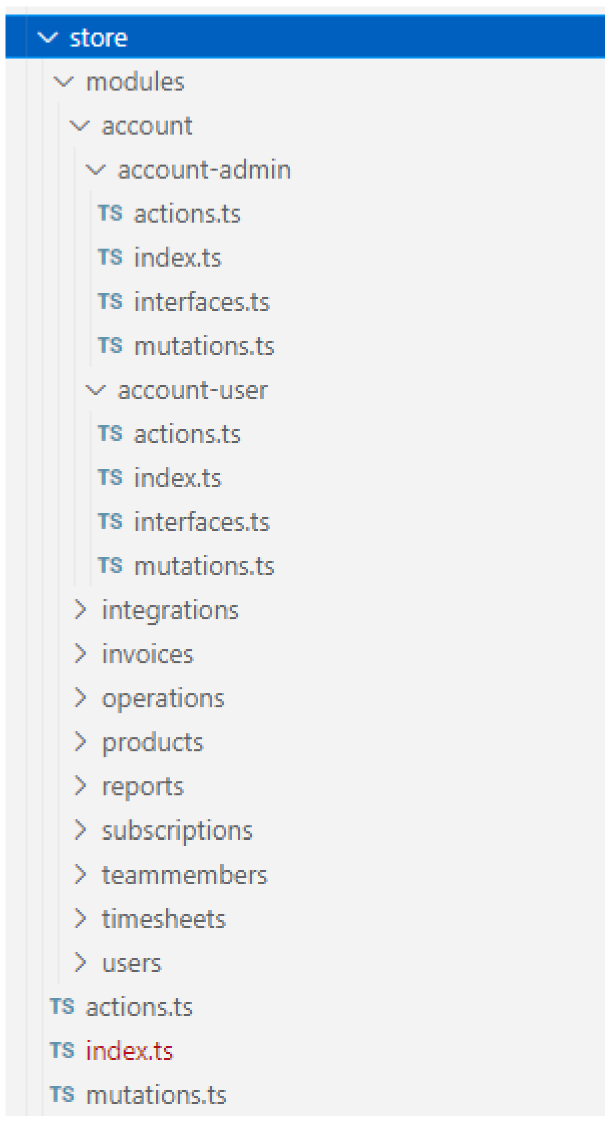
Router

Usually, I keep a single router file, and bind them with the views (pages). However, with increasing project size, you can consider breaking down the router file into more smaller files (by modules). You can then import them into the main index or router file. Here's a tabular example on how you can map the components for each of the module:

View / Component name	Route URL	Description
ProductList	/products	Show list of products
ProductAdd	/products/create	Add a new product
ProductEdit	/products/:id/edit	Edit a product
ProductView	/products/:id	View a product

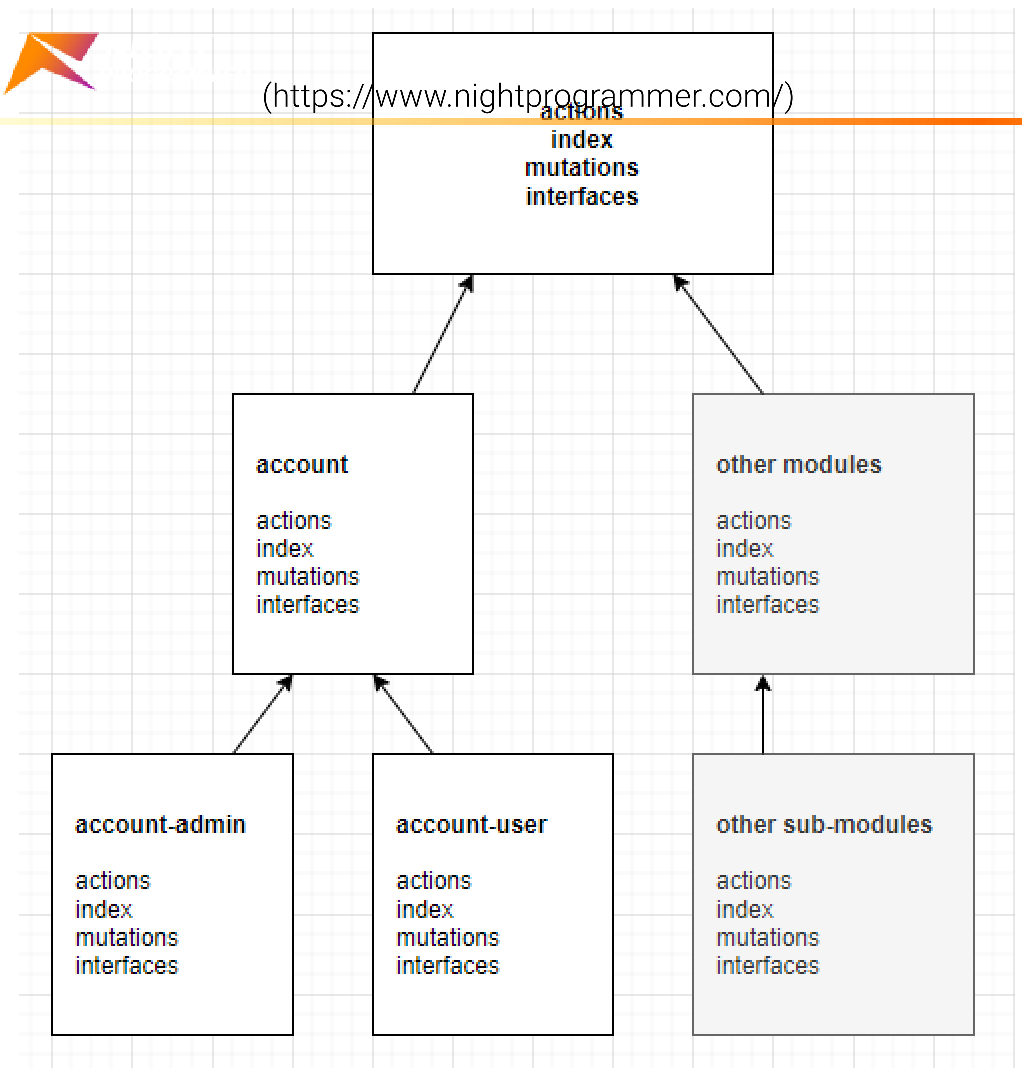
Store

Having a state management library is a must-have tool for any large application. And Vuex is the go-to state manager for Vue.js. Here's a snapshot of how I structure my store modules:



```
store
  modules
    account
      account-admin
        actions.ts
        index.ts
        interfaces.ts
        mutations.ts
      account-user
        actions.ts
        index.ts
        interfaces.ts
        mutations.ts
    integrations
    invoices
    operations
    products
    reports
    subscriptions
    teammembers
    timesheets
    users
  actions.ts
  index.ts
  mutations.ts
```

(https://www.nightprogrammer.com/)



Ask

As you can see diagrammatically, each of the store module is composed of four files:

actions: Here you can write all the async operations, like API calls to the backend server. You can then **commit** and save that value to a state using a mutation. This is how your actions file may look like. Keep in mind, I've removed all the other details from each of the action functions to highlight the main purpose here.

```
import requests from '@/utils/requests';
import { apiProducts } from '@/utils/apiurls';

export const actions = {
    async getProductsList({ commit }) {
        const url = apiProducts;
        const result = await requests.getData(url);
        if (result) {
            commit('setProductsList', result)
        }
    },
    async getProductDetails({ commit }, id) {
        const url = apiProducts + '/' + id;
        const result = await requests.getData(url);
        if (result) {
            commit('setProductDetails', result)
        }
    },
}
```

mutations: Here you can declare and keep all the default state values for that module. You also need to declare the getters for each of the state in this file. You can then use that getter inside a Vue component directly to access a state

```
    productsList: [],
    productDetails: null
}

export const getters = {
    productsList: state => state.productsList,
    productDetails: state => state.productDetails
}

export const mutations = {
    setProductsList(state, data) {
        data.map((product) => state.productsList.push(product));
    },
    setProductDetails(state, data) {
        state.productDetails = data;
    }
}
```

Ask

index: Here you can import state, getters and mutations from the mutations file and export it. You also import actions from the actions file and export it.

```
import { state, getters, mutations } from './mutations';
import { actions } from './actions';


export default ({
    state,
    mutations,
    getters,
    actions,
});
```

interfaces: If you use TypeScript (as I do), you can declare all the interfaces you need in that particular module, here.

```
export interface IproductType {
    id: string,
    type: number
}

export interface IproductInterface {
    id: string,
    title: string,
    unit: string,
    cost: number,
    type: IproductInterface
}
```

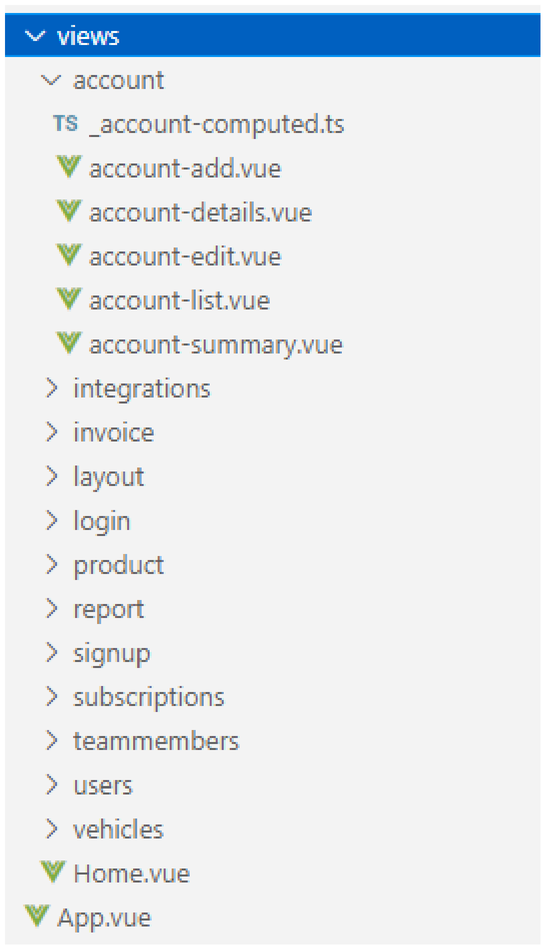
After you've structured each of your module this way, you can then import all the modules with their names in the main index file of the Vuex store. This is how your main index of the Vuex store may look like:

```
import productsModule from '@/store/modules/products/index';
import accountModule from '@/store/modules/account/index';
// other module imports

Vue.use(Vuex);

export default new Vuex.Store({
    state,
    mutations,
    getters,
    actions,
    modules: {
        productsModule,
        accountModule,
        // other modules
    }
});
```

Views



```
views
  account
    TS _account-computed.ts
    account-add.vue
    account-details.vue
    account-edit.vue
    account-list.vue
    account-summary.vue
  integrations
  invoice
  layout
  login
  product
  report
  signup
  subscriptions
  teammembers
  users
  vehicles
  Home.vue
App.vue
```

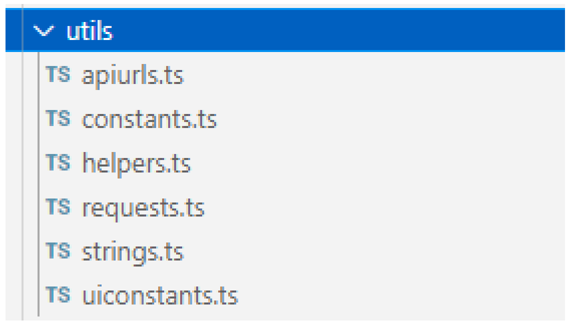
Views are basically the pages which you can directly map with each of the routes. You can create a view by using one or more components. The main views for most of the modules are often list, add, view, and update. I like to keep the nomenclature as **<parent-name><child-name><child-child-name>** and so on. You will also see that I have a file named **_account-computed.ts**. This is where I keep all the getters and setters for that module (account). With a huge extendible project, you'll soon find yourself scrolling endlessly to use getters and setters otherwise. Here's a sample code of that file:

(https://www.nightprogrammer.com/)

```
            get() {
                return this.$store.getters.accountDetails
            },
            set(newVal) {
                this.$store.state.accountModule.accountDetails = newVal
            }
        },
        accountsLinked: {
            get() {
                return this.$store.getters.accountsLinked
            },
            set(newVal) {
                this.$store.state.accountModule.accountsLinked = newVal
            }
        },
        // other getters and setters
    }
}
```

You can then import this file as a mixin in whichever vue component you want to use it.

Utils



It’s impossible to work on a large app rapidly, without having sufficient utility library and methods. These are some of the few utility files I create in almost every Vue project. Let’s have a look at each of them one by one:

apiurls: This is where you can keep all your API endpoint URLs. This saves the hassle of having to change endpoints everywhere in your Vue project whenever there’s an endpoint change.

```
//account
export const apiAccount = '/account';
export const apiLogin = `/${apiAccount}/access/authorize`;
export const apiRefreshToken = `/${apiAccount}/access/renew`;
export const apiPasswordReset = '/account/access/reset';

// users
export const apiUsers = '/users';

// products
export const apiProducts = '/products';

// others
```

constants: The name says it all. This is where you keep all the constants. Here’s a sample:

(https://www.nightprogrammer.com/)

```
export const defaultBusinessLatitudeConstant = 53.95;
export const defaultBusinessLongitudeConstant = -1.08;
export const defaultMapZoom = 12;
export const defaultPageOffsetSize = 0;
export const defaultPageLimitSize = 100;
//and more
```

helpers: You can define all the most required utility functions here. For instance, date time formatter, date time convertor (UTC), floating point number with fixed decimal points, regex validators, save cookie, get cookie, log information, etc.

requests: This is where you write the CRUD methods (GET, POST, PUT, DELETE, etc) using axios for all the asynchronous API calls you to make from the actions file. I'll write a separate article on this.

Ask

strings: Here you can keep all the strings used in the project. Import the string constants wherever you need in the project. This will help you maintain the DRY principle as in case of constants, whenever there is a repetitive usage. Here's a sample code:

```
// products
export const deleteProductWarning = 'Are you sure you want to delete this
export const productCreated = 'New product added successfully';
export const productNotCreated = 'Failed to add new product';
export const productNotUpdated = 'Failed to update product';
export const productUpdated = 'Product updated Successfully';
export const productDeleted = 'Product deleted Successfully';
export const productNotDeleted = 'Product not deleted';
export const productZeroStateHeading = 'You do not have any products';
export const productZeroState = 'Click on the "add product" button to star
export const productListDisabled = "You're not allowed to view the product
```

uiconstants: This is where you can keep all the user interface related constants like the maximum height of some list or the success message box color of some notification.

```
export const successColor = '#182879';
export const failureColor = '#e65a5a';
export const brandColor = '#182879';

export const defaultScrollViewHeight = 700;
export const defaultListViewVH = 0.75;
export const defaultMaxAlertBoxWidth = 400;
export const defaultMaxAlertBoxHeight = 300;
```

(https://www.nightprogrammer.com/)

Previous Post

How to auto-lowercase user input in Vue.js?

(https://www.nightprogrammer.com/vue-js/automatically-lowercase-user-input-in-vue-js/)

Next Post

10 Best Vue.js UI Component Libraries / Frameworks 2022

(https://www.nightprogrammer.com/vue-js/vue-ui-component-libraries-frameworks-2022/)

You Might Also Like

How to register a component locally in Vue.js | Example (https://www.nightprogrammer.com/vue-js/register-component-locally-in-vuejs-example/)

Time duration rounded quarterly payroll format in JavaScript | Example (https://www.nightprogrammer.com/javascript/time-duration-rounded-payroll-in-javascript-example/)

How to lazy load images on scroll in Vue 3 | Example (https://www.nightprogrammer.com/vue-3/how-to-lazy-load-images-in-vue-3-example/)

This Post Has 7 Comments

prashant says:

January 7, 2022 at 7:44 am (https://www.nightprogrammer.com/vue-js/architect-structure-large-scale-vue-js-application-example/#comment-151)

Hello Gautam, your article is good but kindly add github repository link.

Gautam says:

January 7, 2022 at 4:47 pm (https://www.nightprogrammer.com/vue-js/architect-structure-large-scale-vue-js-application-example/#comment-154)

Hi Prashant,
Thanks for the read. It'll take me a while to share the structure on Git (expect a few days), as I'll have to remove all the contents/links of any 3rd party libraries/assets which might lead to violation of privacy policies. If you don't want to wait, you could generate a project using "@vue/cli" and use "vue create project-name" to generate the basic boilerplates. Once done, you could follow along the article as this article is an extended / enhanced version of the structure provided by @vue/cli.

prashant says:

January 7, 2022 at 7:45 am (https://www.nightprogrammer.com/vue-js/architect-structure-large-scale-vue-js-application-example/#comment-152)

Hello Gautam, kindly add github repository link.

Shefali Gupta says:

January 7, 2022 at 4:37 pm (https://www.nightprogrammer.com/vue-js/architect-structure-large-scale-vue-js-application-example/#comment-153)

Hi, thanks for the amazing article! I'm working on an e-commerce project ,and we're using Element Ui. So far everything's good but with increasing number of products, I think I'll probably need a better structure soon. I got the folder structure generated by Vue CLI, do u think it's enough for future application-example/#comment-155)

(https://www.nightprogrammer.com/)

Hello Shefali, thanks for the read. It's possible to scale a project on any given structure. However, what differs is the amount of time taken by developers to maintain those projects. The more modular, atomic your project structure is, the easier it will be to maintain it in the long run. The more your code base grows, the more you'll start seeing the benefits. And for E-commerce apps, an extendible structure is highly recommended. While the template generated by Vue CLI is good enough for small-medium scale apps, I'd definitely recommend you to take a step ahead and breakdown the directories in a modular and reusable way.

santosh says:

January 10, 2022 at 6:14 pm (https://www.nightprogrammer.com/vue-js/architect-structure-large-scale-vue-js-application-example/#comment-156)

Thanks for the amazing article! Will definitely try this structure, I'm waiting for the CSS structure

Ask

Olivia S says:

July 14, 2022 at 8:49 pm (https://www.nightprogrammer.com/vue-js/architect-structure-large-scale-vue-js-application-example/#comment-369)

Splendid structure! Will definitely try to implement it in my next Vue project.

Looking for something?

About us (/about/)

Contact us (/contact/)

Privacy policy (/privacy-policy/)

Have something to say?

Drop us a mail at admin@nightprogrammer.com

NIGHT PROGRAMMER

made with ❤ and coffee