N technical implementation

In the whole project, we adopted the development method of separating the front end and the back end, in this way, we can focus on the unilateral learning and development of front-end technology or back-end technology during the development process to improve efficiency. During the test the source of the BUG can also be found more quickly and sent to the front-end or back-end engineers for modification respectively. Focusing on the future, by separating the front end and the back end, the real front end and the back end can be decoupled to reduce the server load caused by the increase of traffic in the future. If the server goes down due to excessive business requirements, there will be no significant impact on the front-end pages. Meanwhile, due to asynchronous loading, no matter how complex the front-end pages are, the response speed of the server will not be affected. In this section, we are going to introduce Front End and Back End.

N.1 Front End:

We divided the front end into two parts, customs’ interface and employees’ interface. Both of them were built based on VUE framework. When we coding the most different between VUE and traditional HTML+JavaScript+CSS model is in VUE frameworks we put all these three parts into one file. This method is called Single file VUE componentized development model[[1]](#footnote-0). The code in a VUE file look like the Figure N.1. 

Figure N.1 Simple code for VUE file

N.1.1 Responsive programming by using VUEx:

Responsive programming is one of the features of the VUE framework, there is a folder called store, a file called store.js can be found in this folder. This file is the key part to realize the responsive programming. The code in this file can be divided into three part: view(Mapping state to the view declarative), state(Data sources that drive applications), actions(Responds to a state change caused by user input on the view), it looks like the Appendix[N] Figure[N] (store.js). All state changes happened on store, will be managed on store’s action part, this method called Centralized State Management. Because of using Centralized State Management, we can find which kind of mutation will happen and how. When an error happened, we will also have a Log of what happened lead to BUG.

N.1.2 Route Manager plug-in Components: VUE-Router

Our web page was designed to take into account the switching of different pages and the switching of sub-pages within the page. The VUE framework provides a good solution to this problem. The VUE-Router plug-in makes it easy to build a jump between pages. A file called router.js can be found in root directory of our front end folder , we show the code in this file in Appendix[N] Figure[N](router.js). This file store all the link between pages in our website. If we want use the link, we just need to import the router in router.js like the code in Figure N.2 below.



Figure N.2 The sample code of use router

Vue-Router also used in Detection platform switching different screen adaptation schemes, such as PC and mobile switching, by calling the “replace” method to replace the display of the page, you can automatically display different layouts[[2]](#footnote-1). The following Figure N.2(login.vue 247-249) is an example of using the replace method.

N.1.3 Component Development by using iView

Another features of VUE framework is component development. Its implementation is mainly embodied in the use of iView component library. The advantages of iView component library are:

1.high quality, rich function, delicate and beautiful UI.

2.friendly API which makes components insert on web more easily.

3.An official document of every detail.

4.Single file VUE componentized development model

5.Developed base on npm+webpack+babel, Compatible ES2015[[3]](#footnote-2).

Let’s talk about the advantage No.4: Single file VUE componentized development model. Look at Appendix[N] Figure[N](same as N.1), we can find out there are three labels: template, JavaScript, style. In a single VUE file, template is responsible for templates, JavaScript is responsible for logic, and style is responsible for styles. The idea behind this is that a single file component corresponds to a functional component, and the template, style, and business logic for that component all adopt the idea of nearby maintenance. From the perspective of component reusability and late maintainability, this concept greatly improves the development efficiency of componentization. VUE single files are neither JavaScript, HTML, nor CSS files. In our development process, we think of a page as a large component composed of multiple components.

Next, we'll show you how these components collaborate to accomplish some representative system functions[[4]](#footnote-3).

N.1.4 Login and Registration page

We think of the entire login page as a large component, with a smaller component in the center that provides a location where the company logo is displayed to the customer, as well as an input box for the customer to enter the user name and password. Below is the login and registration button, which triggers an event that interacts with the back end and receives the return value from the back end. In this way, the user name and password confirmation and user rights distinction. The implementation code for this section is as Appendix[N] Figure[N](login.vue input part 82-107).The company logo is showed by <img/> label. Text is displayed in this format: {{ $t("message.Welcome") }}[[5]](#footnote-4), the input box is realized by <Input/> label, <Button/> label has supported the “Login” and “Register” button. All primary style was written on style part of the VUE file, only especial style was controlled by label itself.

A click event called modal1 is used on the registration button, and by triggering the event we can provide the user with an interface to fill in the registration information. An push method will be called by router[[6]](#footnote-5) to find the correct registration page and show it to user. On the registration page, we also make use of the same components to provide input boxes and text compared to Login page, but the difference is that we add some function to make sure the user will enter the data that we want. As the user enters the registration information, a method named “riRule” is called. This method is mainly used to detect whether the necessary information is correctly filled in, for example: is the password entered? Is the password input the same two times before and after? E-mail address format is correct, etc. If the information entered does not meet the requirements, the appropriate prompt is returned，the sample implementation is shown in Figure N.3(Login.vue 216-243).



Figure N.3 sample implementation of riRule

N.1.5 Home page

In our design, the home page is divided into two components, the top navigation bar and the home page content. The top navigation bar is controlled by the “index.vue” file, and the home page content is controlled by the “home.vue” file.

Let's start with the top navigation bar, which is implemented using the <menu> label, and each of the navigation bar items is treated as a <menuitem/> and then numbered separately to identify click listening events. See Appendix[N] Figure[N](index.vue 89-166) for specific implementation.We also use a conditional rendering method called “v-if”. The feature of this method is that when the value returned by “v-if” is true, the component will be displayed, and otherwise the component will be hidden. In the implementation of the navigation bar, we set a Boolean variable named “loginFlage” with an initial value of false. By placing the classification of the components in the navigation bar in two different <div/> labels with different “v-if” return values, we have implemented the option display and hiding of the navigation bar in the unlogged-in and logged-in states, such as the login button and the log-out button. We put the login button in the <div/> labels where the “v-if” return value is not equal to the “loginFlage”, so that the login button is displayed when it is not logged in, and the log-out is reversed. The switch language button, which is used in both cases, placed under both <div/> labels.The following Figure N.4(index.vue 158-164) shows the navigation bar component detail code that is displayed

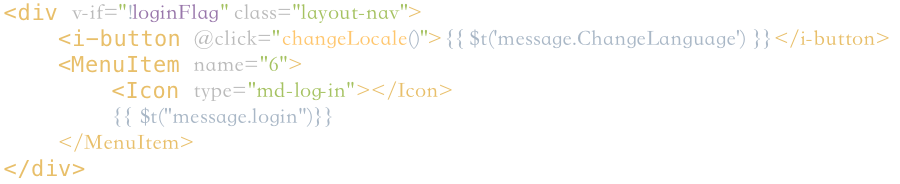


Figure N.4 Home page navigation bar component detail code

in the unlogged-in state.The only different between logged-in state and unlogged-in state is the “v-if” is equal to “loginFlage” or not. Considering that users need to input large text description when using the system, we introduce a “interest-quill-editor”[[7]](#footnote-6) rich text editor based on VUE framework. It supports title, paragraph, reference, superscript, font size and other commonly used article editing patterns, perfect to meet the user needs of our vision. Detailed implementation is shown in the Figure N.5(create-article.vue 22).



Figure N.5 interest-quill-editor implementation

Next, we'll introduce some of the technique that are used only in the content section of the home page.On the front page of our design, there is a component that shows the picture, which switches the content displayed in it on time to achieve our goal of dynamically presenting our product category to our customers. This technology, called Walking Lamp, is implemented through <Carousel/> label. The “autoplay” property in the tag causes the content displayed in the component to switch automatically, while the loop property causes the stored content in the component to start over automatically after broadcast. Detailed implementation is shown in the Figure N.6(home.vue 4-10)



Figure N.6 Detailed implementation of Carousel

N.1.6 Console

The supply technology used in the process of implementing the employee console is almost the same as the front page. The input box is provided by input, and the side navigation bar is designed to provide rich text editing by menu,interest-quill-editor. In order to manage the user's questions and applications, the employee side adds the form component to realize the user's question list and so on. This function is realized by calling the form tag. Considering the excessive number of entries in the table, 10 pieces of data can be displayed on a single page by setting the “pageSize” variable. At the same time, a page flipping button is added. The concrete implementation is shown in the Appendix[N] Figure[N](form.vue 29-65)and Appendix[N] Figure[N](form.vue 232-235)

1. We will introduce it in N.1.2 Component Development [↑](#footnote-ref-0)
2. On the mobile side of the adaptation scheme please see N.N.N [↑](#footnote-ref-1)
3. Also called ECMAScript 6 which is a standard of JavaScript released on 2015. [↑](#footnote-ref-2)
4. Features page implementations that are not mentioned are much the same. [↑](#footnote-ref-3)
5. This will be introduced in N.1.N vue-i18n internationalization (different languages).Same as below [↑](#footnote-ref-4)
6. For more information, see N.1.2 Route Manager plug-in components: VUE-Router. Same as below. [↑](#footnote-ref-5)
7. The original version is called vue-quill-editor,. We're referring to a better rewrite. [↑](#footnote-ref-6)