Next level vue

* Interceptor: là một lớp ở giữa, được thực thi trước khi api được gửi đên server, và thực thi trước khi nhận được response trả về từ phía server. Ứng dụng : ta có thể load thành progress khi bắt đầu gọi api, và tắt thanh progress sau khi gọi api thành công. Lọc các response trả về, thêm tham số vào request trước khi gửi đi ví dụ như thêm token vào các api.

Text

Description automatically generated

**Vue router.**

Như đã biết, với các component của vue, ta sẽ có 8 lifecycle hook.

* BeforeCreate : khởi tạo data, event
* Created: tạo instance thành công
* BeforeMount: compile template & element
* Mounted: dữ liệu được mount vào template thành công
* BeforeUpdate
* Updated
* BeforeDestroy
* Destroyed

Tuy nhiên khi sử dụng vue router, chúng ta sẽ có thêm 3 vue router hook nữa đó là

* BeforeRouteEnter (to, form, next)
* BeforeRouteUpdate (to, form, next)
* BeforeRouteLeave (to, form, next)

A screenshot of a cell phone

Description automatically generated

To là route chuẩn bị được chuyển hướng

From là route hiện tại

Next() là function để chạy sang route mới. Ta có thể truyền thêm các tham số vào cho hàm next().

**Vue route guard**

* Global router guards:

được áp dụng cho toàn bộ các route.

Text

Description automatically generated

Thứ tự chạy của router hook

A screenshot of a cell phone

Description automatically generated

* Per route guard

Bên cạnh global guard, ta có thể viết từng guard cho các route cụ thể.

A screenshot of a cell phone

Description automatically generated

**Truyền dữ liệu từ router guard sang component không sử dụng vuex.**

Luồng ban đầu:

Khi component created, dispatch action fetchEvent, dữ liêu từ api truyền về sẽ cập nhật lại state trên vuex. Do state của vuex được cập nhật nên component sẽ rerender lại.

Thay vì phải cập nhật vuex, r vuex cập nhật component, ta có thể truyền trực tiếp dữ liệu từ guard của router vào props

Text

Description automatically generated

Đầu tiên, ta set thuộc tính **props = true**

Tiếp theo, trong hàm beforeEnter: **do sử dụng method chaining của promise (.then().then().catch() ) nên ở action fetchEvent ta sẽ cần trả về 1 promise.**

**Text

Description automatically generated**

Đồng thời, để sử dụng được kết quả của method chaining trong promise, ta cũng cần return thêm dữ liệu event.

Graphical user interface, text, application

Description automatically generated

Quay trở lại với route beforeEnter, do dữ liệu được trả về từ bên action, nên ta có thể sử dụng biến event và đưa nó vào trong params của routeTo.

Cuối cùng, ở phần view, ta chỉ cần khai báo props event là có thể dùng được

Text

Description automatically generated

**Base component**

Trong trường hợp chúng ta muốn ta ra các base component như input, select. Thay vì dùng trước tiếp các thẻ html như bên dưới.

A picture containing text

Description automatically generated

Ta có thể tách ra được 1 base input như sau:

Text

Description automatically generated

ở đây, ta sẽ sử dụng v-model ở component cha, và bên component con sẽ sử dụng **this.$emit(‘input’)** để cập nhật dữ liệu của v-model cha. Chúng ta cũng có thể bind các thuộc tính hay class vào component con với v-bind và inheritAttrs

Cách sử dụng

Text

Description automatically generated

Nhìn lại class baseinput đã xây dựng, ta sẽ thấy nhờ câu lệnh **v-bind =”$attrs“**



Nên các thuộc tính như type, placeholder cũng được bind vào trong thẻ input. Còn do class không được bind nên nó sẽ nằm ở thẻ root của component. Hãy thử inspect thẻ base input trên web ta sẽ tháy

Graphical user interface, text, website

Description automatically generated

Ở đây, class=field mặc định sẽ nằm ở thẻ root của base component. Còn các thuộc tính khác như type, placeholder do chúng ta đã bind vào thẻ input, đồng thời set thuộc tính **inheritAttrs = false**, nên các thuộc tính type, placeholder sẽ không có xuất hiện ở thẻ root nữa.



Trong trường hợp, **inheritAttrs = true**  kết quả sẽ như hình bên dưới, cả thẻ div và thẻ input đều sẽ có các thuọc tính type, placeholder như này

**Vuelidate**

**Yarn add vuelidate**

Graphical user interface, text, application

Description automatically generated

Import thư viện vuelidate vào file main.js

Sử dụng:



Ta sẽ có 1 thẻ input như trên. Tiếp theo tiến hành viết các rule cần validate cho thẻ input này

Text

Description automatically generated

**Import rule required và email từ thư viện của vuelidat**e. Đồng thời khai báo thêm 1 thuộc tính **validations** ngang cấp với với hàm data() .

Mở brower và vào tab Vue trong dev tool

**Graphical user interface, application

Description automatically generated**

Ta sẽ thấy ở component example lúc này sẽ có thêm 1 **object $v** do vuelidate sinh ra.

A picture containing text

Description automatically generated

Trong object $v này, ta sẽ thấy có một số thuộc tính như trên, và đặc biệt là thuộc tính email, chính là thuộc tính mà ta đã khai báo trong validations ở trên.

Ta sẽ thấy, thuộc tính $invalid mặc định bằng true, thuộc tính email = true, và thuộc tính required = false.

Tương ứng với các rule mà nó đang được validate. Khi tất cả các rule validate = true thì $invalid = true. Dựa vào các gíá trị này mà ta có thể show là lỗi hiển thị cho người dùng biết .

A picture containing graphical user interface

Description automatically generated

Cách để truy cập vào giá trị validate cũng rát đơn giản, như chúng ta truy cập 1 object $v bình thường.

Tuy nhiên, nếu validate như này thì mỗi khi người dùng mở trang web lên, các lỗi sẽ được validate và hiện lên ngay lập tức khiên cho trải nghiệm người dùng không được tốt. Do đó, chúng ta cần phải kiểm tra và hiển thị lỗi khi người dùng bắt đầu thao tác với ô input hoặc khi người dùng submit form.

Graphical user interface, text

Description automatically generated

Ta sẽ băt sự kiện blur đối với thẻ input, khi người dùng click vào ô input, ta sẽ đánh dấu thẻ input nãy đã bị touch => $dirty = true.

Đồng thời ta cũng có 1 thuộc tính $error = $dirty + $invalid ( true + false = false, true + true = true, false + false = false);

DO vậy, chỉ cần kiểm tra biến $error là ta có thể biết được người dùng đã tương tác với thẻ input chưa đồng thời giá trị người dùng nhập có gây ra lỗi hay ko.

Tuy nhiên, chúng ta mới chỉ đang kiểm tra validate khi người dùng tương tác với input thì sẽ hiện ra lỗi, còn trong trường hợp người dùng không tương tác với input và submit form luôn, thì rõ ràng lúc này các. Lỗi không thể hiện lên được.

A picture containing text

Description automatically generated

Để ý lại thì ta sẽ thấy, ở object $v, cũng có các thuộc tính $dirty, $error giống như input email. Đây chính là các giao của các validate con của toàn bộ form. Do đó, khi submit form, chúng ta cũng cần kiểm tra giá trị biến $v.$error để đảm bảo tất cả giá trị của form đã được validate chính xác

Text

Description automatically generated

Validate đối với các base component .

Text

Description automatically generated

Tương tự đối với các thuộc tính như type, placholder được class con kế thừa từ class cha thông qua v-bind=”$attribute”. Ta cũng có thể gán sự kiện @blur, @click cho base component thông qua **v-on=”$listeners” .**



Conflict trong base input

Các bước thực hiện cũng tương tự đối với BaseSelect. Tuy nhiên, khi viết đoạn code y hệt đối với BaseInput.

Graphical user interface, text

Description automatically generated

Lỗi ở đây là do sự conflict giữa các sữ kiện trên thẻ input.

Ta thấy thẻ input có 1 sự kiện @input để update giá trị của v-model bind vào. Đây là sự kiện do chúng ta định nghĩa. Tuy nhiên, khi sử dụng v-model, vuejs cũng tự định nghĩa sự kiện @input.

Graphical user interface, text, application, chat or text message

Description automatically generated

Đoạn code bên dưới diễn giải những gì v-model làm.

Do người dùng tự định nghĩa @input, và v-model cũng định nghĩa @input. Cho nên, khi ta sử dụng **v-on=”$listeners” thì sẽ bị conflict ở sự kiện này.**

Cách xử lý ở đây là ta sẽ phải ghi đè sự kiện input do người dùng định nghĩa trong object $listener bằng computed:

Text

Description automatically generated



**Mixin (giống trait bên laravel)**

So sánh 2 base component BaseInput và BaseSelect ta sẽ thấy các đoạn code và biến bị trùng lặp, như vậy ta có thể tách ra được thành mixin

Text

Description automatically generated



Filter( filter trong app chỉnh sửa ảnh – dùng để biến đổi dữ liệu): giống như transformer trong laravel, khi dữ liệu đi qua các filter, dữ liệu sẽ được biến đổi theo thư tự của các filter. Đối với các biến đổi phức tạp và hay được dùng đi dùng lại ở nhiều nơi thì chúng ta sẽ dùng đến filter, còn trong trường hợp đơn giản chúng ta chỉ cần sử dụng method, computed là được.