**COMPOSITION API:**

Esto es lo que hace poderoso a Vue, tomo lo mejor de react y lo mejor de angular

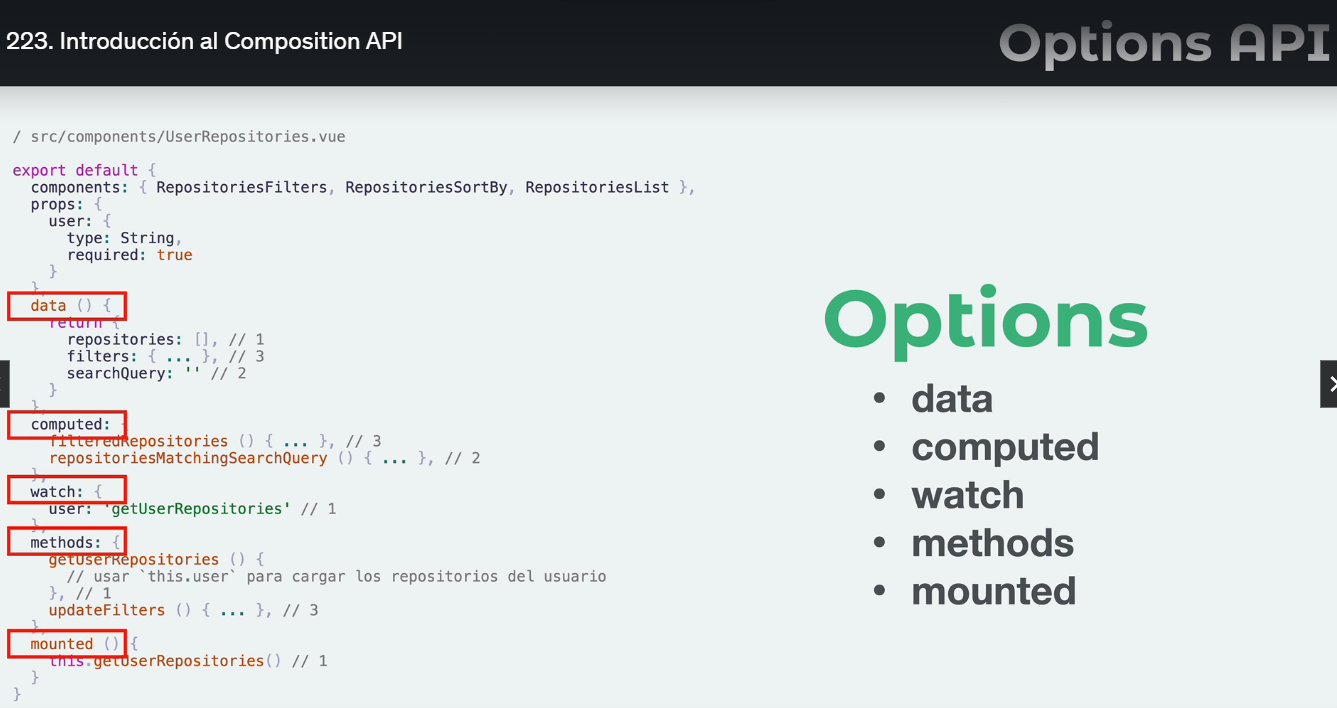
En el options api se usan mixings que es un objeto que tiene las opciones que necesitamos compartir, pero no se usan mucho porque generan problemas. Es como que con un mixin podes agarrar funciones de otro componente y usarlas.

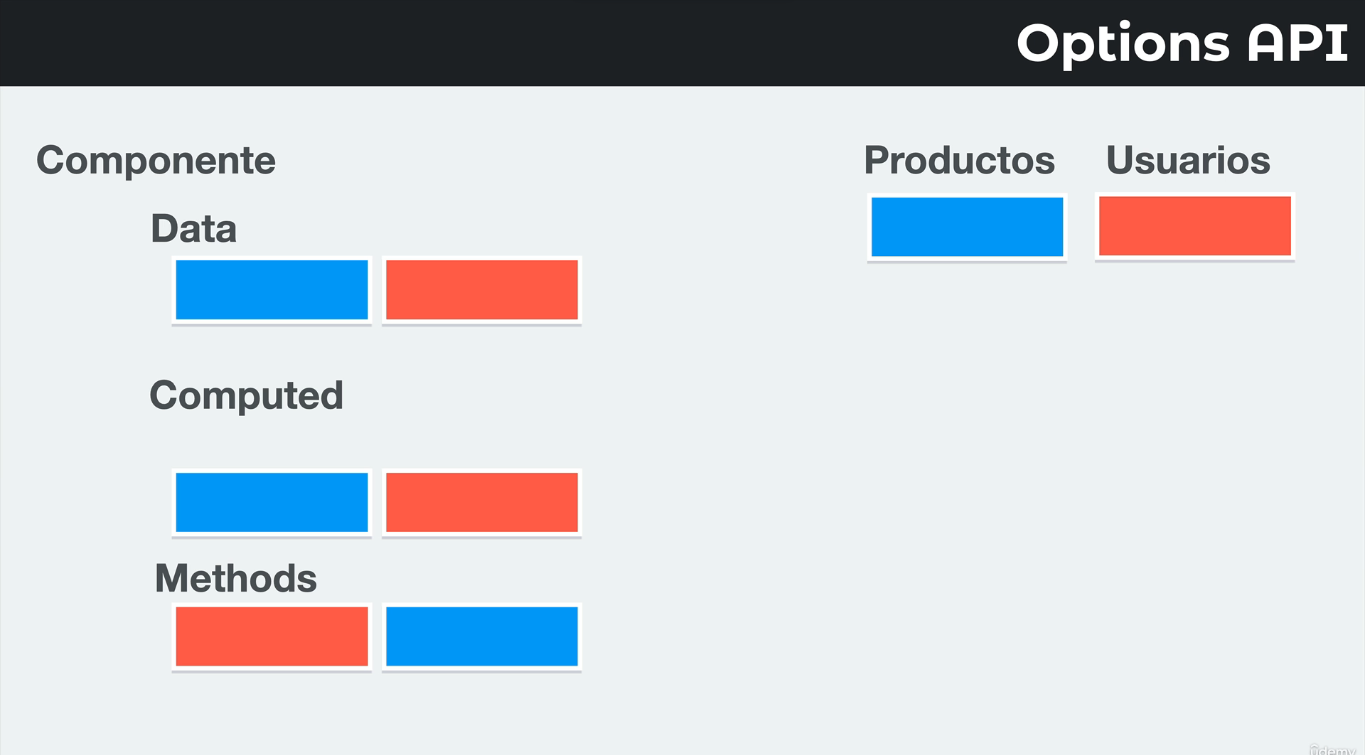
En composition api también tenemos todo el ciclo de vida de vue. Dentro del setup nosotros especificamos todo lo importante del componente

**Introducción composition api:**



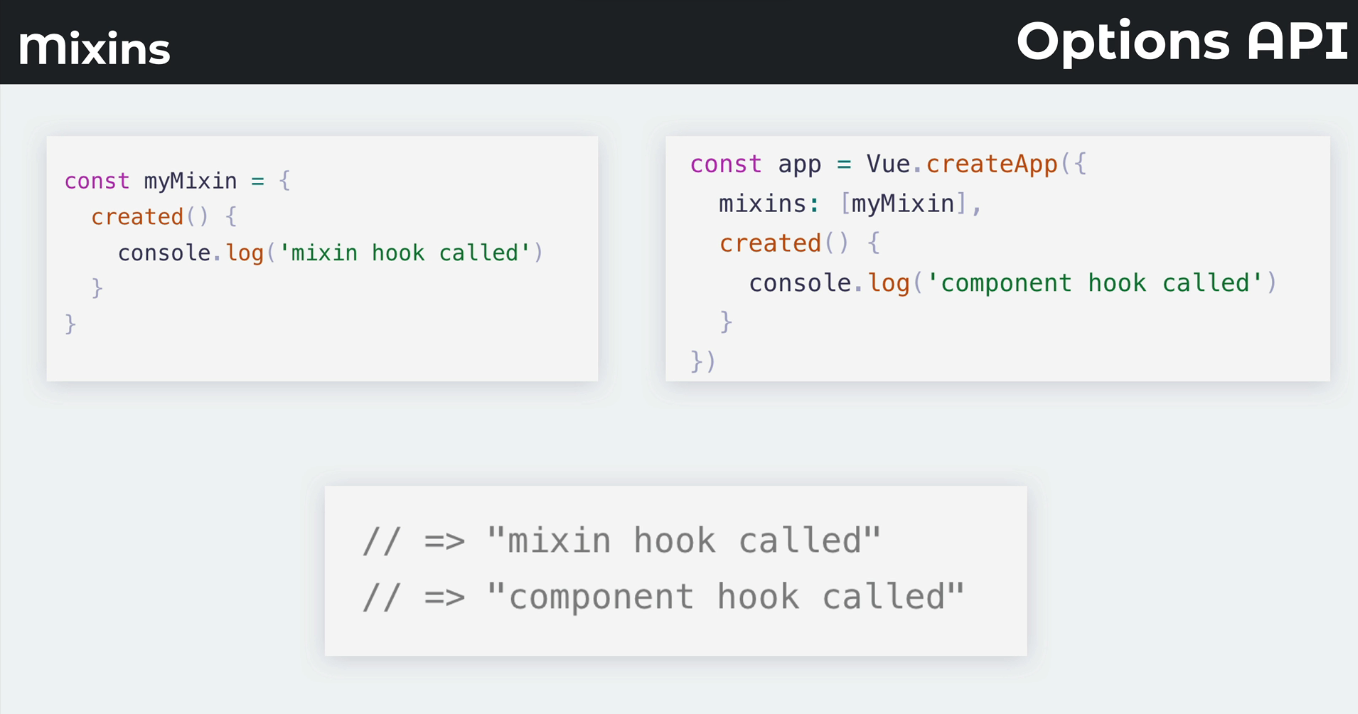
El composition api es opcional.



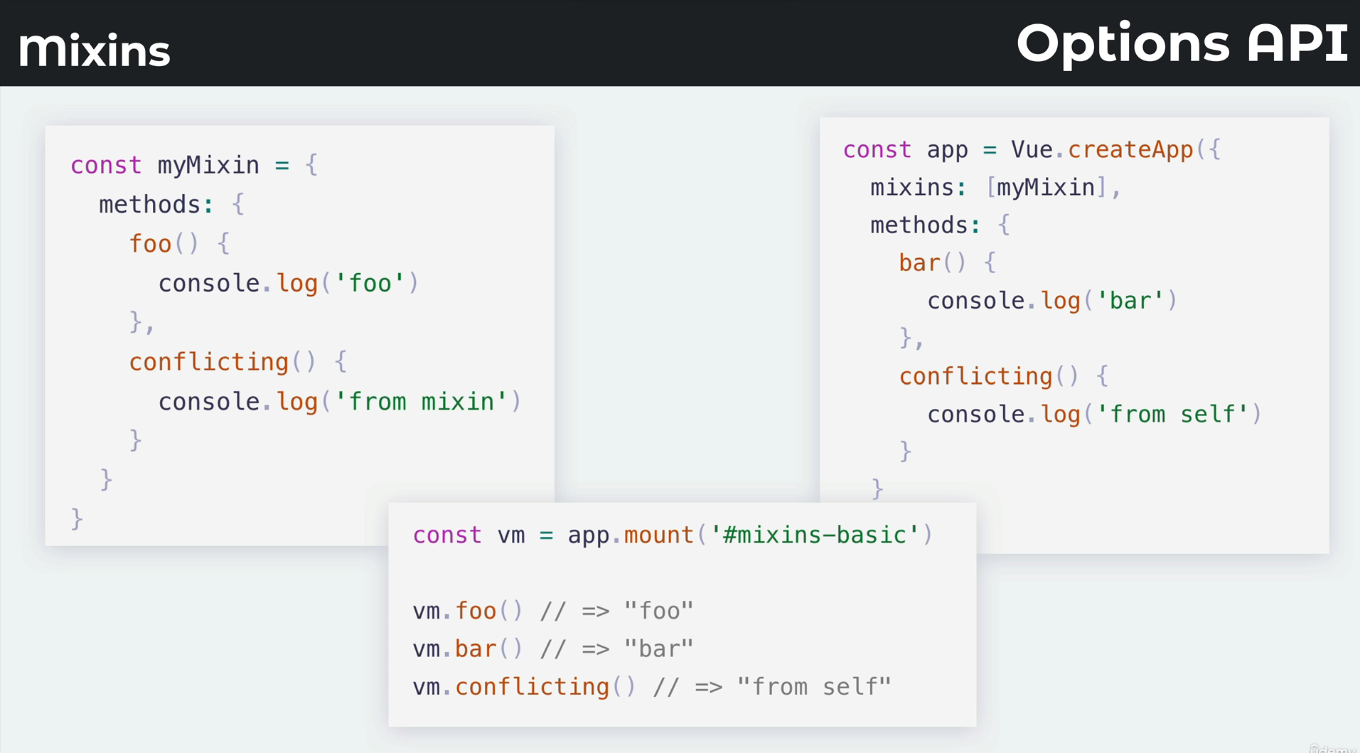


Queda difícil de seguir se mezcla todo con el options api

El mixing es una solución que se daba cuando se quería reutilizar cierto código en otros componentes

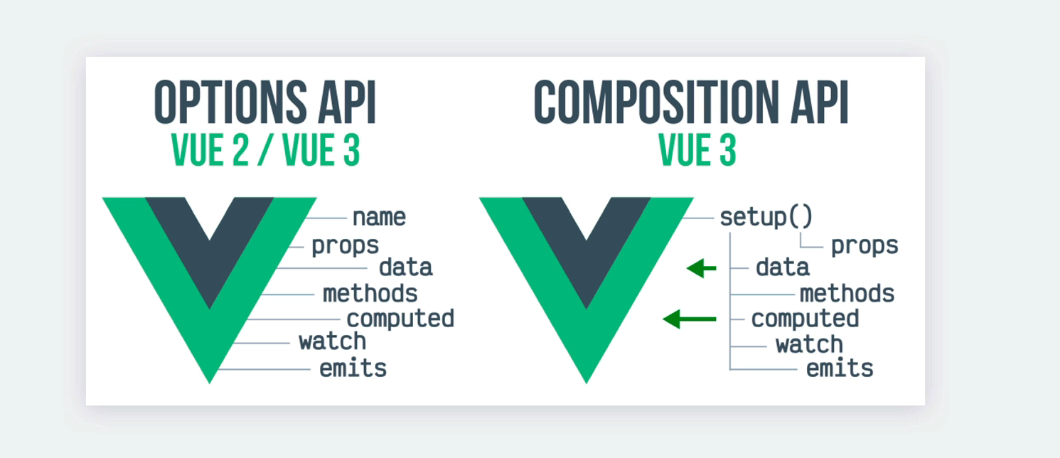


Hay problemas si en el mixin tenes una función que se llama igual en el componente, ahí va a tener prioridad la del componente



En tanto a ciclo de vida primero se ejecutan los del mixing y después los del componente





El setup se ejecuta antes que todo, antes que el created, mounted etc. Se ejecuta cuando el componente esta siendo creado.

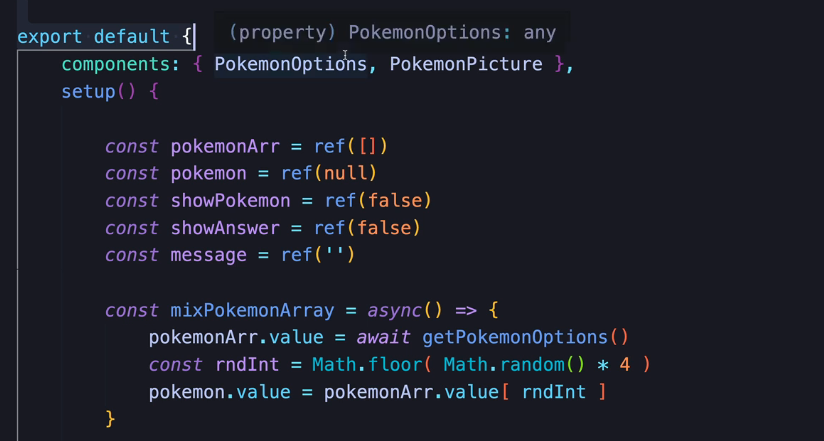
No se usa la referencia al this en composition.

Ejemplo:

Hace un import de ref de vue

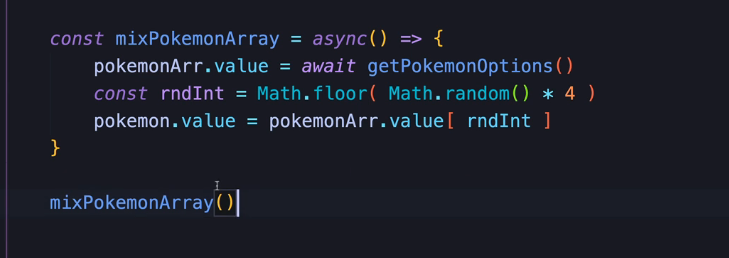


Después el export default los componentes y el setup



Esas constantes son lo que antes hubiera ido en la data. El ref lo que hace es crear una envoluta de lo que sea que se mande adentro,

La función sepuede llamara ahí mismo



Por ejemplo ahí en pokemonArr.value, nosotros para poder acceder al valor de la referencia tenemos que usar ese .value

En el return vamos a poner todo lo que nosotros necesitamos exponer en el template



Ahora todo eso se puede resumir en algo así, pero todavía no explicó como

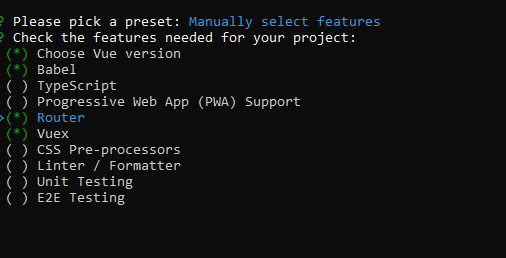


Ese usePokemon() es similar a los hooks de react acá se llaman composables

**Inicio de proyecto composition API:**

Abrimos cdm de Windows en la carpeta a instalar, corremos:

Vue créate nombre



Usamos vue 3

Lo del router no

In dedicated config files

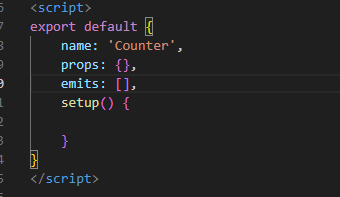
Npm run serve para correrlo

Borramos el componente de HelloWorld

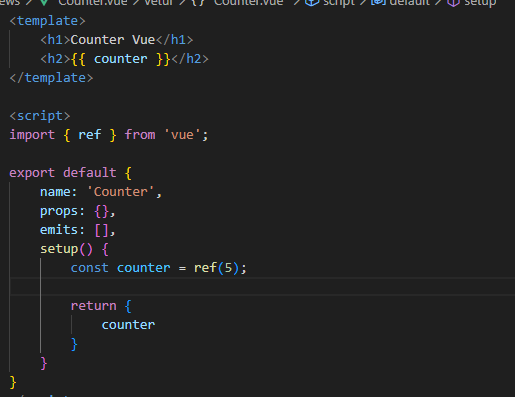
En el Home borramos la referencia del HelloWorld

RECORDAR DE MODIFICAR EL NPM RUN SERVE, ESTA EN LAS PRIMERAS HOJAS

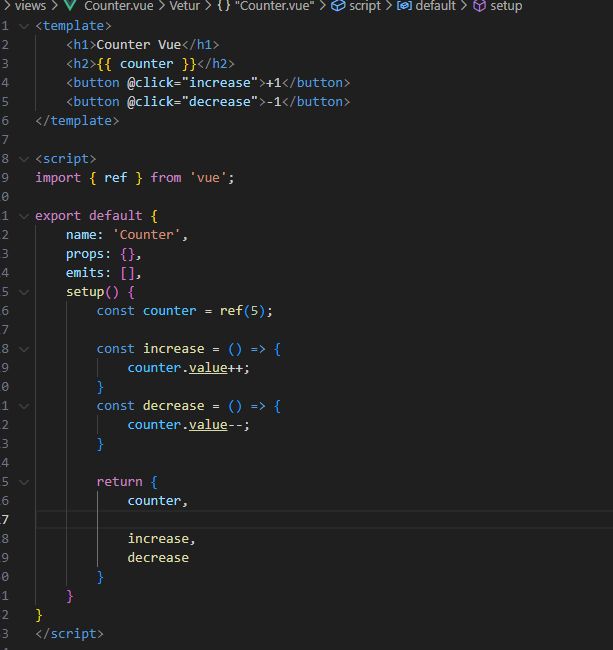
Creamos un nuevo componente, lo agregamos a las rutas y al header del app

Por lo general estas son las cosas que quedan por fuera del setup   


En vezde definir las variables en data lo que hacemos es definirlo como una constante, y para poder usarlo en el html lo tenemos que poner en el return si o si



Para funciones también creamos constantes que van a ser funciones y también se retornan



Ahora no es tan comun ver las funciones definidas así si lo único que van a hacer esas funciones es usarlas en el html

Es mejor hacerlo directamente en el return



**Creando primer composable:**

Los composables son funciones que nos regresa objetos reactivos, métodos estados, etc.

Creamos una nueva carpeta en el src que se llame composables

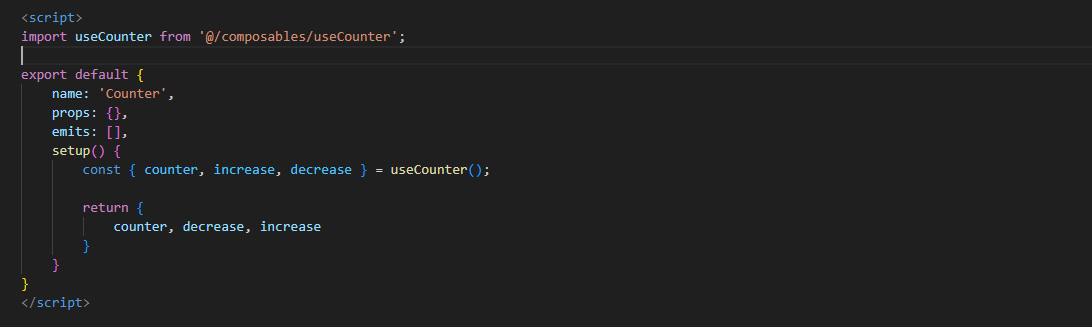
Creamos este file, se usa esa nomenclatura con el use antes



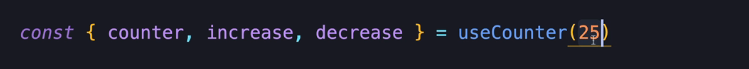
Ahí creamos una función y la exportamos, dentro podemos poner la misma lógica que teníamos en el componente no hay que olvidarse de importar el ref



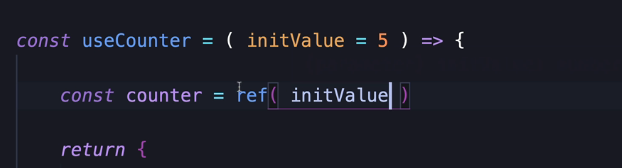
Después en el componente podemos importar el composable, y podemos desestructurar todos los valores que necesitemos del useCounter y después los retornamos si es necesario usar en el html



Lo que se puede hacer con esto, que no se puede en vue2 es enviar argumentos

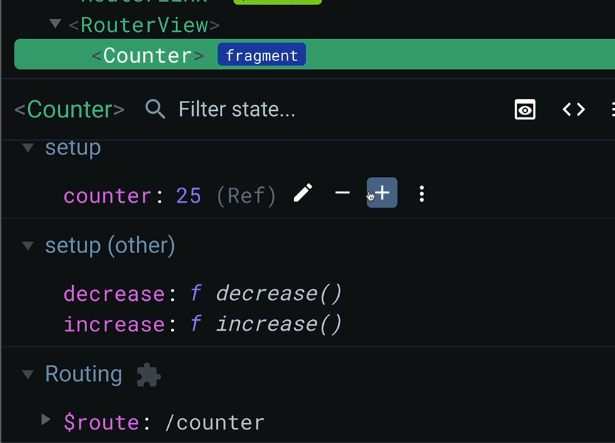


Por ejemplo si ahí queremos enviar el 25 lo va a poder recibir en el composable



Si no se recibe nada el valor por defecto va a ser 5

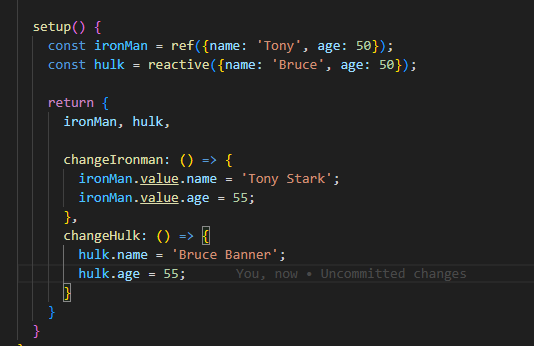
**VueDevTools composition API:**

****

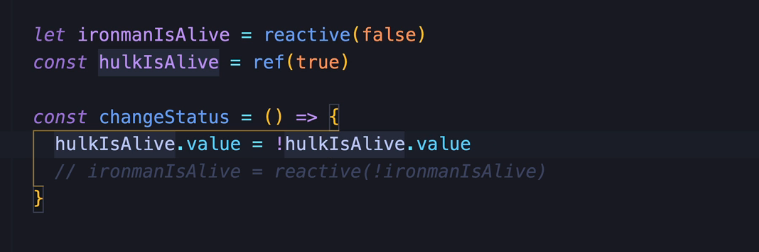
Usar la versión ultima que tome vue3

**Ref vs reactive:**

Con el reactive no se usa el .value para acceder al valor de las variables y **reactive solo funciona con objetos**



En este caso si funciona, pero si queres hacer algo así no va a funcionar

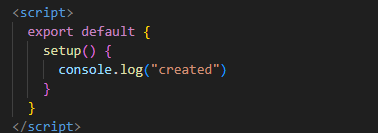


**Lifecycle hooks:**

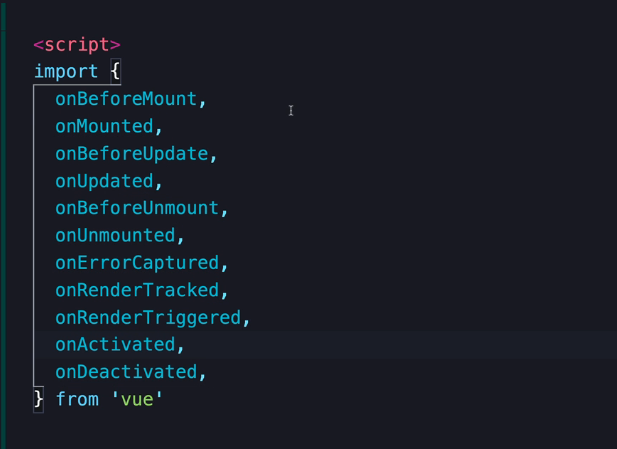
<https://vuejs.org/api/composition-api-lifecycle.html>

No hay que mezclar los ciclos de vida del options api con los del composition api

El beforeCreate y el créate se llaman cuando el setup es ejecutado entonces no se usan.

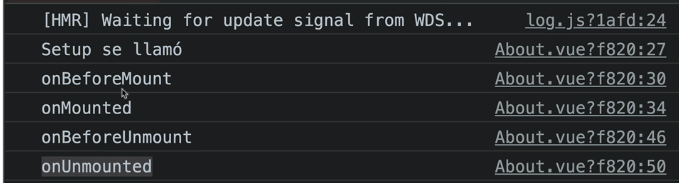


Hay que importar el ciclo de vida que haya que usar

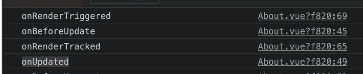


Y se llaman como un callback:





onRenderTriggered es cuando se ejecuta un evento que va a hacer que se cambie algo del html



**Router keep-alive**

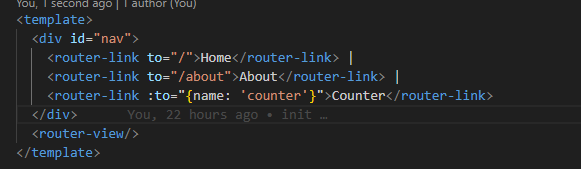
Si vos por ejemplo, entras al counter haces un cambio después vas a about y después a counter de nuevo es como que se resetea



Como se puede mantener ese estado sin necesidad del local storage?

<https://router.vuejs.org/guide/migration/#scrollbehavior-changes>

En el app ya tenemos un router-view que se encarga de renderizar cada una de las vistas dependiendo de la url



El v-slot todavía no lo explico pero básicamente toma el componente que coincide con la ruta y lo muestra en <component> . esto va en el app.vue



Con esto cada vez que cambiemos de ruta se van a mantener los cambios que se hicieron en un componente. En este caso para cualquier componente

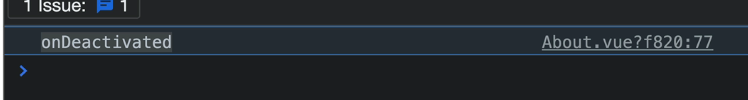
Como hace esto Vue?



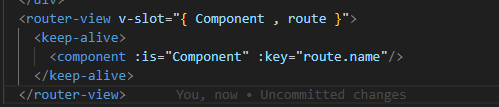
Cuando seleccionas uno sale del keep alive y después cuando te vas de ese componente lo vuelve a meter ahí



Ahí se ejecuta el onDeactivated no el Unmounted , el keep alive previene que se destruya

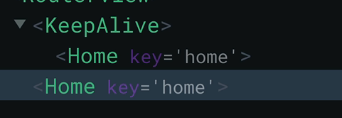


También se puede poner aunque no es obligatorio el key



Que eso va a tomar el name de las rutas del route

Y lo podemos ver en el vue devtool



Ayuda a Vue a saber cual es el componente que debe estar cambiando.

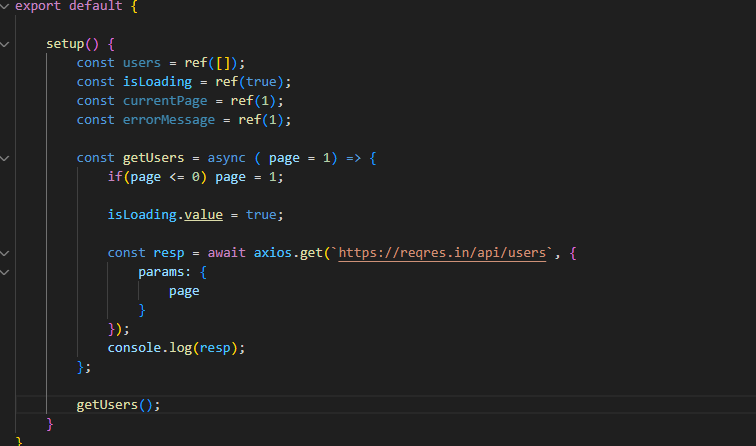
**Peticiones Http:**

Creamos una nueva vista llamada Users.vue, la agregamos a las routes y en el app

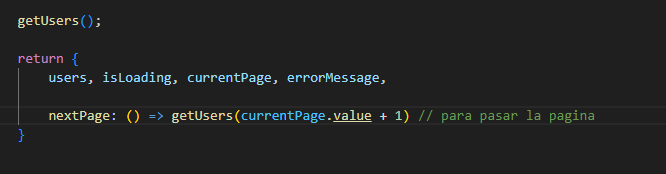
Api ficticia

<https://reqres.in/>

Por ejemplo si tenes que hacer una llamada con axios cuando se carga el componente, se define y se llama ahí mismo en el setup



En el return podemos crear una función que llame a otra función del setup

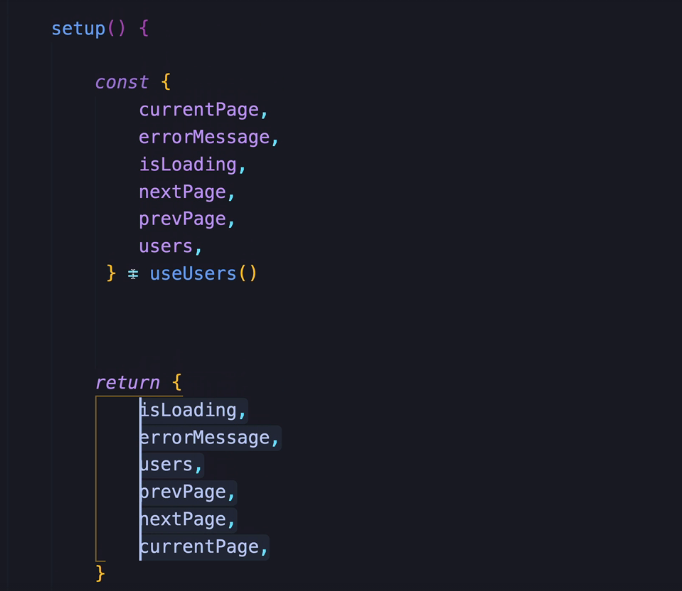


Para usar las variables en el html no hace falta usar el .value



**Importante!** No hay que hacer asíncrono una función composable

Con los composables a veces queda medio redundantes tener la desestructuración y después el return con los mismos elementos

****

Se podría hacer así: en este caso desestructuramos todo lo que el useUsers regresa



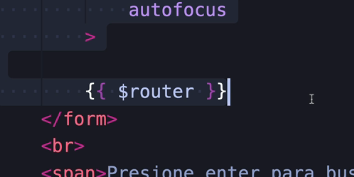
Pero bueno, así no sabes de donde vienen las funciones problema similar al de los mixins

**Router dentro del setup:**

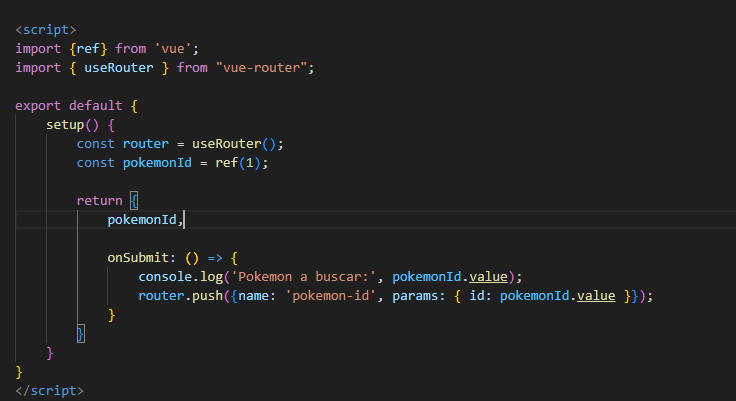
<https://router.vuejs.org/guide/advanced/composition-api.html>

Esto es solo compatible con versiones superiores e iguales a vue router 4

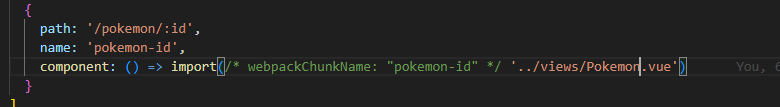
Se podría usar el router directamente en el html sin necesidad de importar nada, esta guardado como global



Para poder usar el router en el script, lo que vamos a tener que hacer es importar useRouter lo guardamos en una constante y ya podemos usar sus métodos como el push que nos sirve como para pasar de una ruta a otra



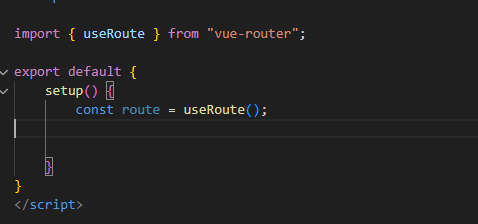
En este caso la ruta de pokemon-id esta recibiendo un parámetro que es el id por eso se lo pasa así



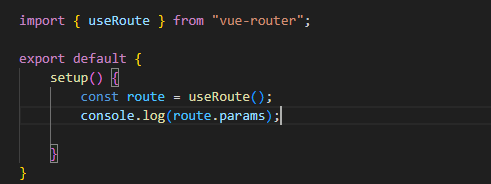
Esto esta mandando los parámetros y la ruta por la url

Ahora veamos como obtener el valor del Id desde la url

Igual que en el anterior vamos a tener que importar useRoute y crear la constante



Podemos acceder a los parámetros con route.params



**Usar refs para focos:**

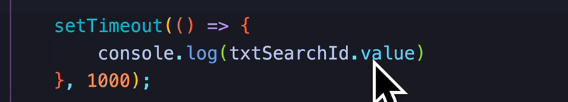
Podemos establecer la referencia en el html como siempre

****

Y tenemos que crear una constante con ese nombre

****

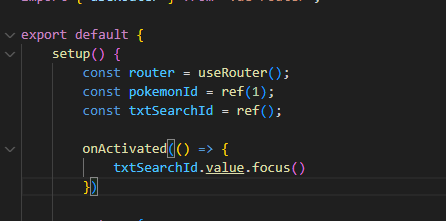
Hay un problema, habría que esperar un segundo para poder obtener el valor de .value de la referencia



Entonces vamos a usar el ciclo de vida onActivated, que se usa ese en vez del onmount por el keep alive

Lo importamos y creamos la función





También en vez de focus podemos usar .select() que eso va a seleccionar también el texto de adentro del input

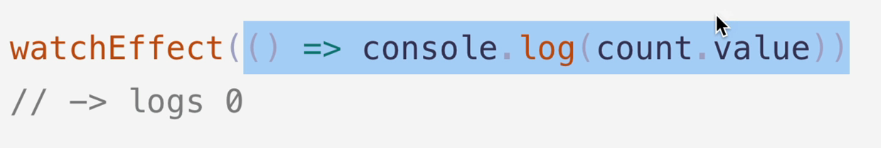
**Observar cambios en objetos reactivos - Watch:**

A veces pasa que no vemos los cambios en los objetos reactivos y tenemos que refrescar la pagina para que se actualice.

Esto pasa porque el componente no se vuelve a montar sino que queda ahí con el keep alive

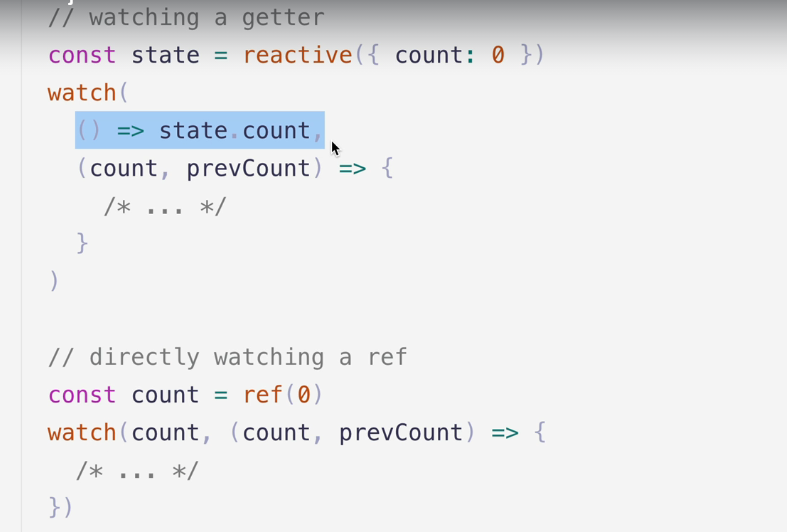
<https://vuejs.org/api/reactivity-core.html#watch>

WatchEffect sabe cual de los elementos reactivos cambió y se vuelve a ejecutar cuando esos cambien

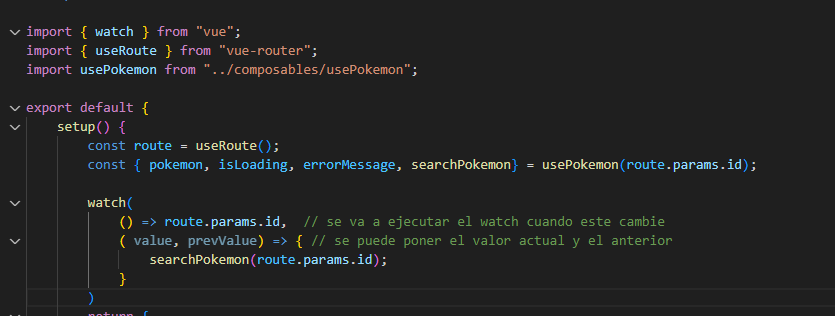


El problema de esto es que no siempre se quiere ejecutar esto

Para esto tenemos el watch y ahí especificamos que queremos estar escuchando, Se puede usar de estas dos maneras



Para usarlo hay que importarlo y en el primer callback va el valor al que queremos que este escuchando, y segundo lo que queremos que se ejecute si se produce algún cambio



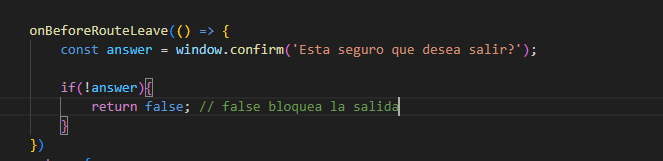
**Bloquear la salida de una ruta:**

Podemos importar este onBeforeRouteLeave



Con esto podemos estar al tanto de cualquier cambio de ruta. Se ejecuta antes de que se cambie de ruta

Dentro del setup ponemos esto:

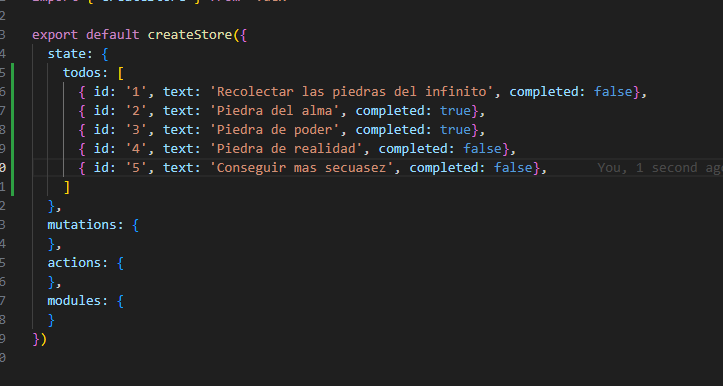


Al retornar false en esta función bloquea la salida y el cambio a otra ruta

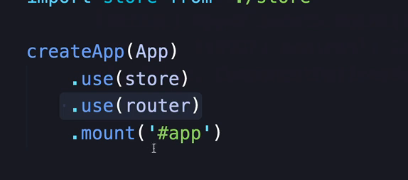
**VueX con compositionAPI:**

Cambian algunas cosas otras se mantienen igual.

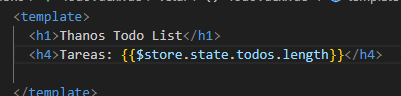
Creamos un state de lista de cosas para hacer



Enel main.js ya como creamos la app con el cli ya se vinculo el archgivo para usar el store



Para acceder al store directo del html



**VueX getters – composition api:**

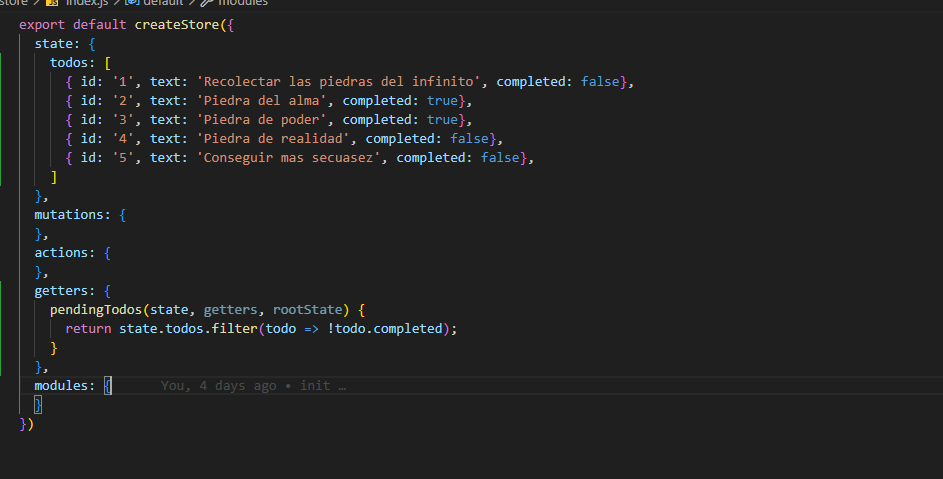
Para poder usar el store en el componente vamos a tener que hacer estas dos importaciones



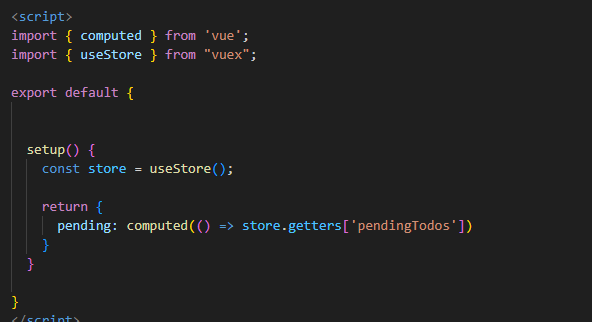
El commit ejecutaba mutaciones

El dispatch para despachar acciones las acciones terminan llamando a la mutación.

En el store creamos un getter que va a traer todos los elementos de la lista que estén pendientes

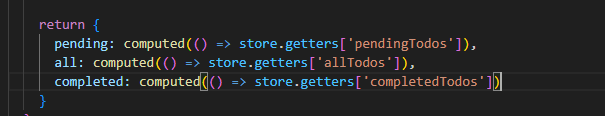


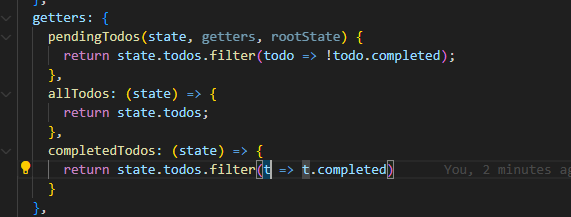
Para usar ese pendingTodos getter desde el componente se hace así y es reactivo, si el state cambia se va a ejecutar la función



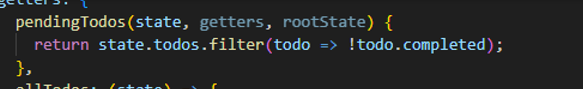
Pending va a guardar un array de los elementos pendientes

Hizo un getter para los completados, pendientes y totales

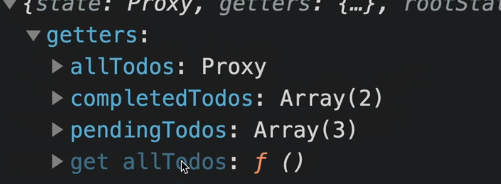




Ahora que onda con estos otros parámetros?

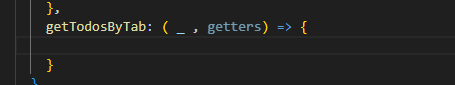


Con el getters tenes acceso a los otros getters que creaste

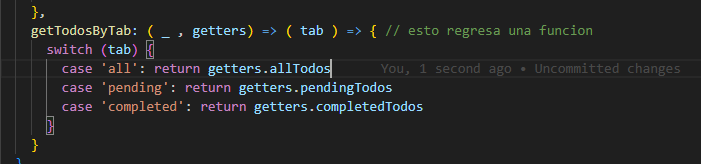


Y state y rootState son distintos, parece que el root state saca lo que hay en el archivo directamente

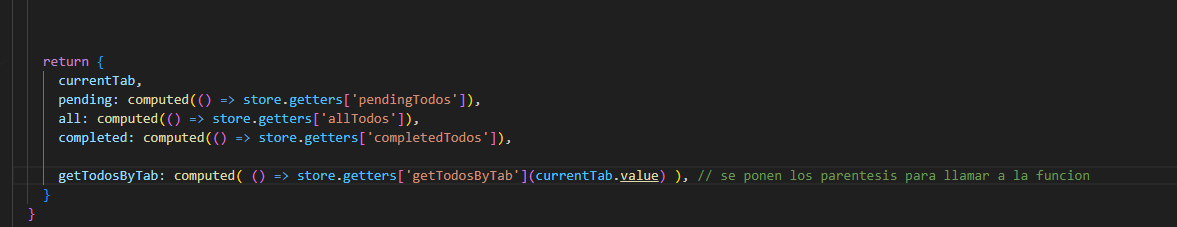
A veces, el primer parámetro no se necesita en este caso el state, por convención se pone un guión bajo para indicar que viene porque es obligatorio pero no se usa



Este getter devuelve una función, esta función retorna un getter que este getter retorna el state con los datos

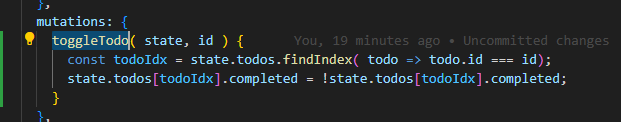


Y se llamaría algo así:



**Mutations composition api:**

La mutación :



La llamada a la mutación, si fuera en el setup

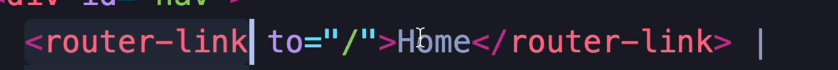


Si fuera directo en el return parece ser que no hace falta usar el cpmúted en este caso



**Introducción a los slots:**

Los slots básicamente son los elementos que se envían dentro de de dos etiquetas de un componente, por ejemplo acá el texto de home sería un slot

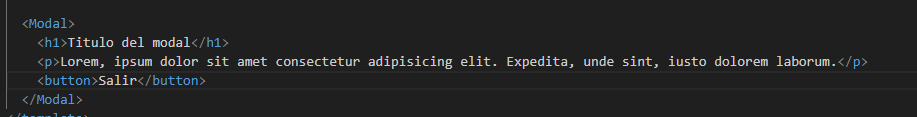




El keep alive es algo que se le envía a un slot del router view . Para que quede claro ese keep alive cae dentro de un slot que esta en router view

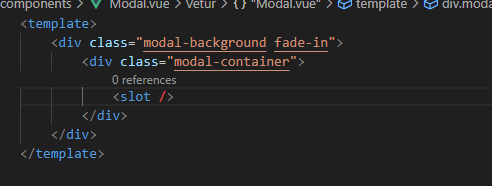
**Slots y NamedSlots:**

Básicamente si nosotros tenemos un componente creado personalizado que es en este caso el modal, nosotros queremos poder mandarle dentro de esta etiqueta mas info como el h1, el p y el button



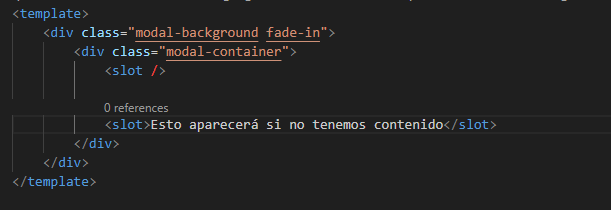
Desde el componente Modal como hacemos para recibir ese html? Con un slot

Si nosotros ponemos una etiqueta slot dentro del html del hijo va a estar mostrando todo lo que se le mande del padre



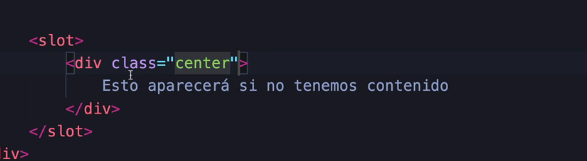


Si no se le manda nada del padre se puede poner otro slot para que muestre un texto



Podemos dejar solamente el segundo, entonces cuando se envie algo va a mostrar lo que se envie si no se envia nada va a mostrar el mensaje

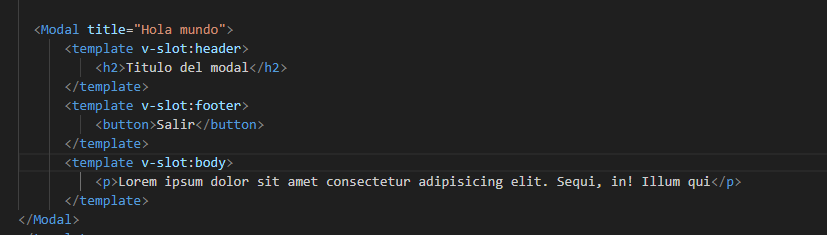
Ósea no haría falta el <slot/>

Si le queremos poner clases habría que crear un div dentro y ponerle a ese la clase  


Nosotros podemos crear varios slots con distintos names,

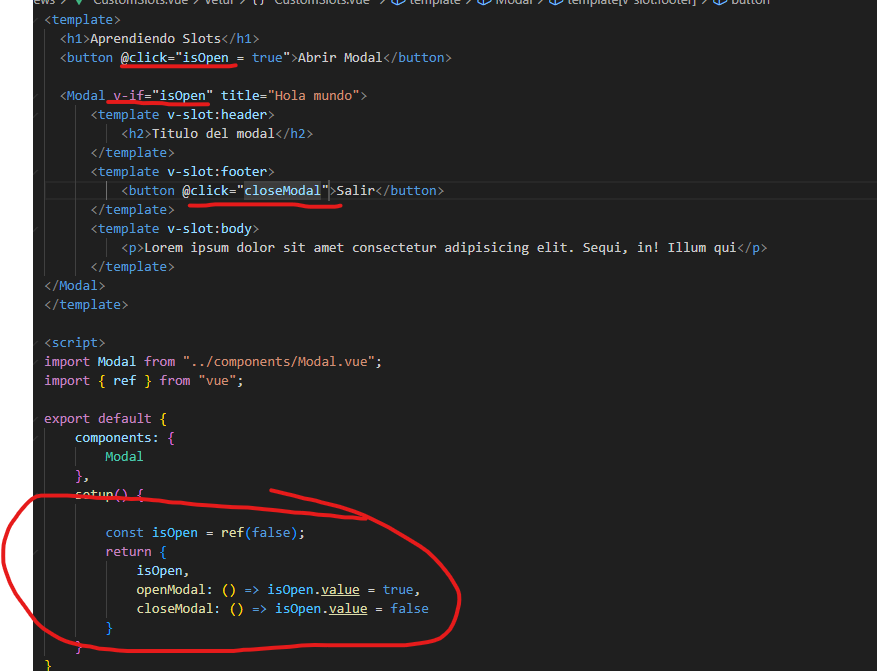


Y después en el padre, dentro del componente hijo le mandas los templates con los v-slot correspondientes

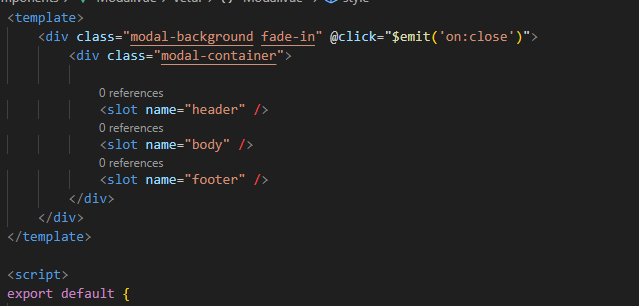


**Funcionalidad del modal con slots:**

Básicamente para abrir y cerrar lo hace desde el padre ya que tiene acceso al botón de salir



Podes emitir eventos del hijo al padre con los emits



Se pueden definir acá abajo también como para que ayude el autocompletado



Así podemos llamarlo en el padre de esta manera

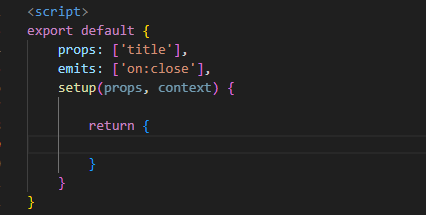


closeModal cambiaba la variable de isOpen a false

**Scoped Slots:**

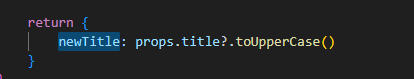
<https://vuejs.org/guide/components/slots.html#dynamic-slot-names>

Supongamos que en un componente hijo recibimos una prop que es title,





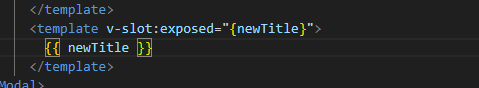
Y en el hijo retornamos esa prop, pero en mayúscula



En ese mismo componente, se va a crear un slot que va a pasarse como “prop” del slot ese valor que recibió

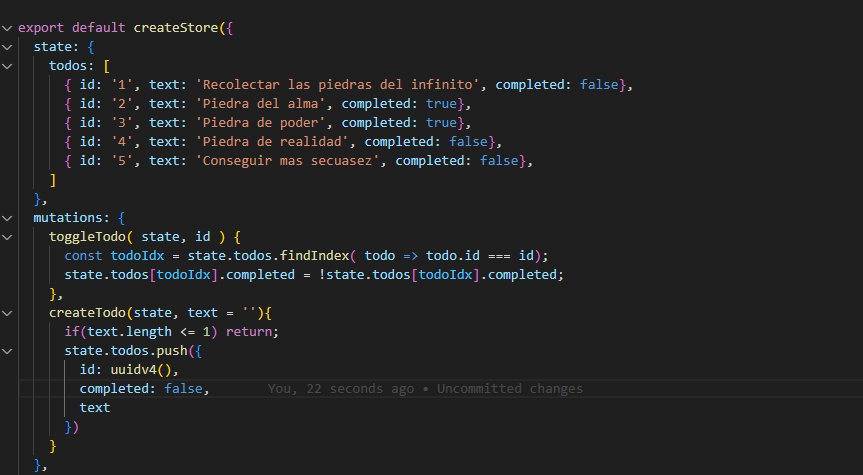


Y desde el padre cuando creamos el contenido para el slot podemos desestructurar todas las props que se le hayan mandado al slot

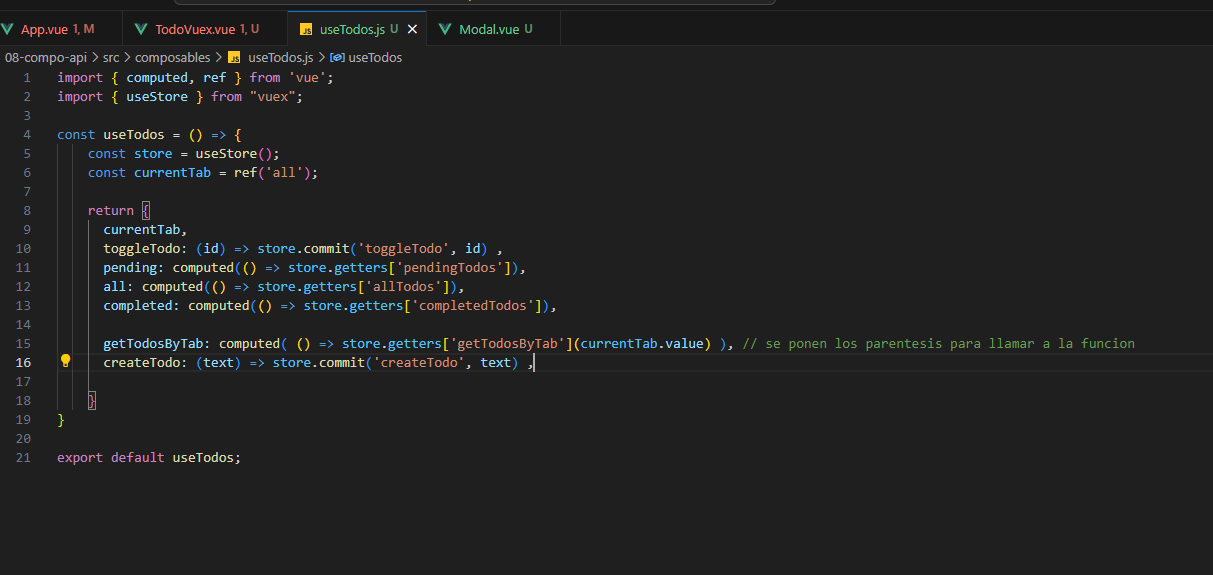


**Crear una nueva tarea en la lista:**

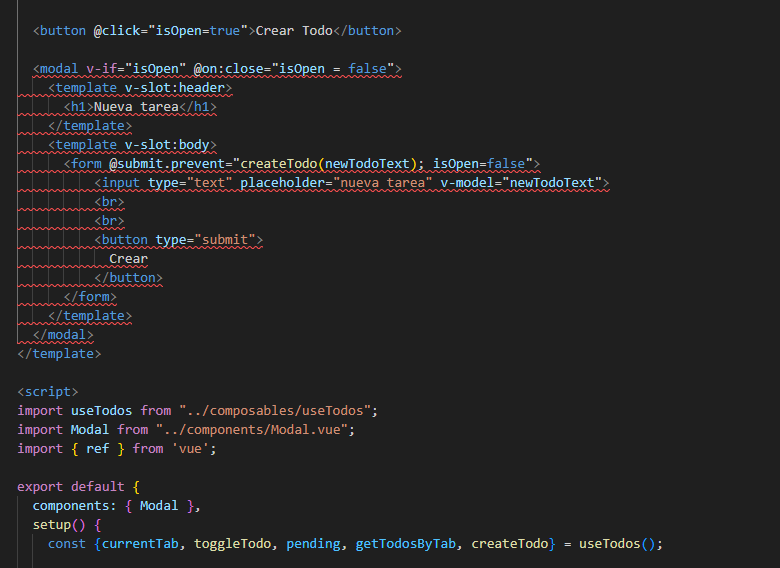
Creamos una mutación para crear un todo



Creamos un créate todo en el composable de todos



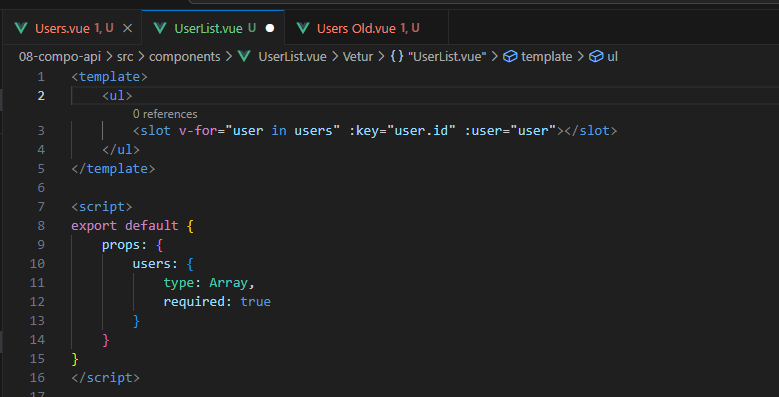
Creamos el componente del modal y llamamos a la función de createtodo



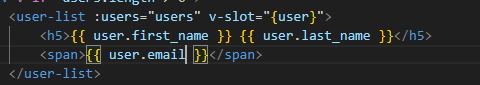


**Scoped Slots aplicado:**

Tenemos un componente de userList con un slot, ese slot recibe del padre los usuario que recorre y el :user=”user te permite usar los elementos desde el padre”

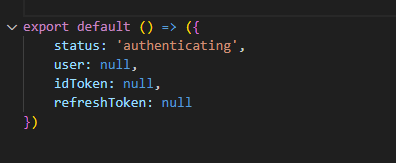


Entonces en el padre podemos usar el componente de esta manera desde el v-slot desestructuramos el user y lo vamos a poder usar a nuestro antojo adentro del componente



**Autenticacion con composition API:**

Creamos el layout para ver el login y el registrar y creamos estos states en el store . Tener en cuenta que este proyecto ya estaba, es el 07-journal



**Registro:**

Toma los datos del form y ejecuta una función

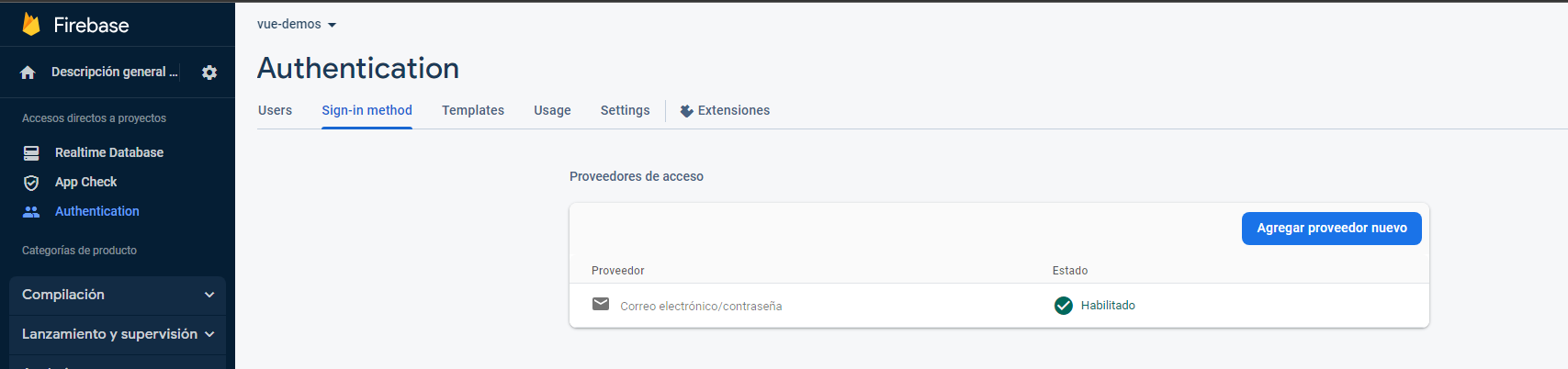


Creamos un composable que va a ser el encargado de registrar el usuario, todavía no se hizo la lógica ese useAuth sería el composable

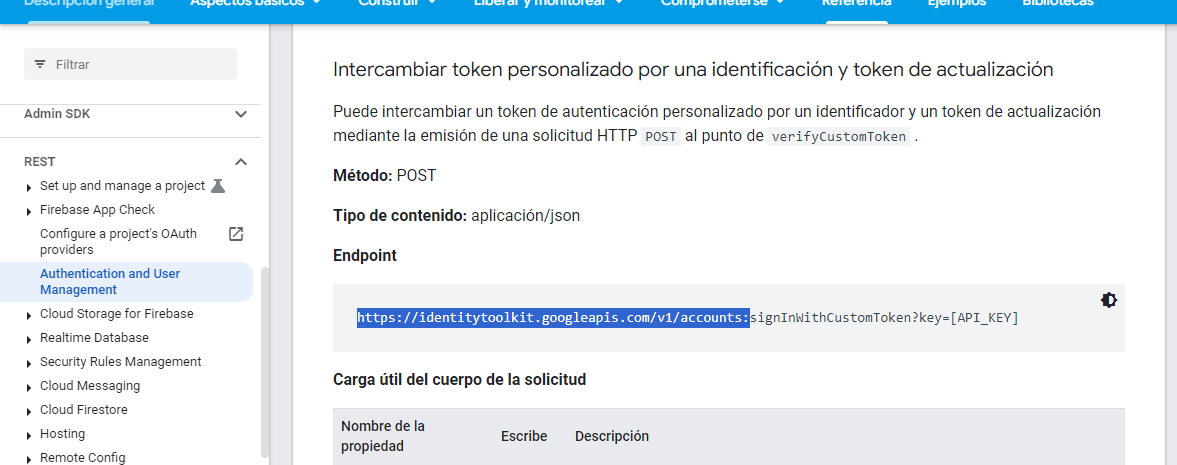


Obtener token de acceso de firebase

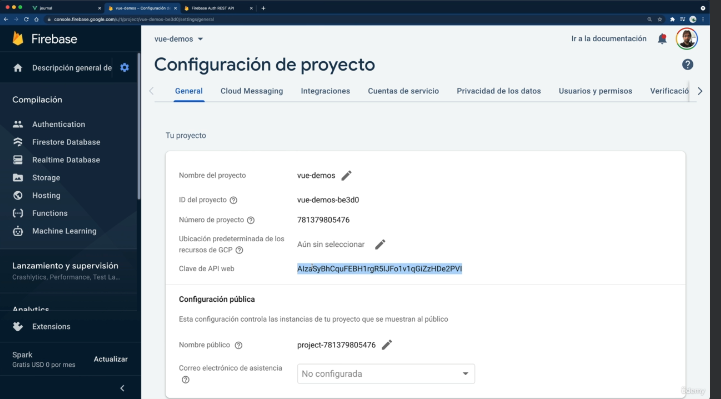
Se viene a la parte de autenticación sig-in methods y se habilita para que sea con mail y password

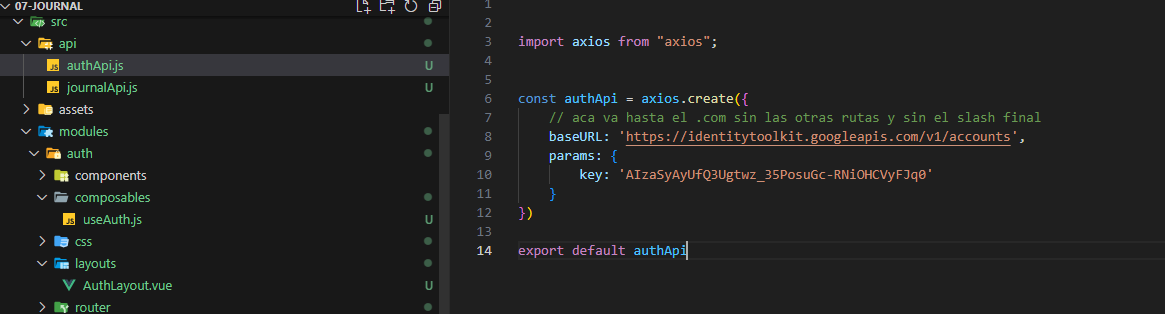


Vamos a usar el endpoint que nos brinda firebase para realizar un registro de un usuario



Y el parámetro va a ser la clave de api de firebase





Hacemos una acción desde el store para realizar la llamada   

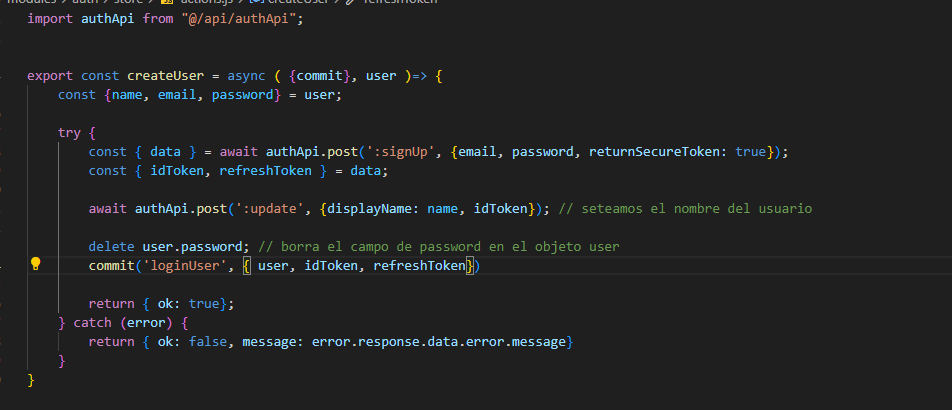

Actualizamos el nombre de usuario



**Mutation : LoginUser:**



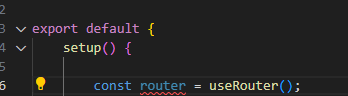
Hacemos el commit en la acción

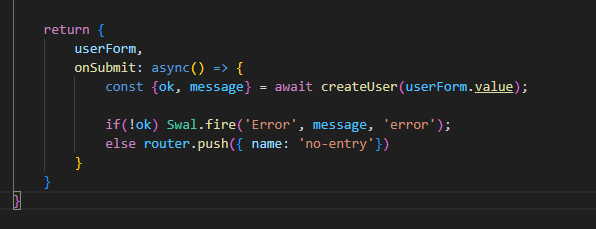


**Mensaje de error o navegación al crear usuario:**

En composition API para poder usar el router podemos usar un composable que es de vue router

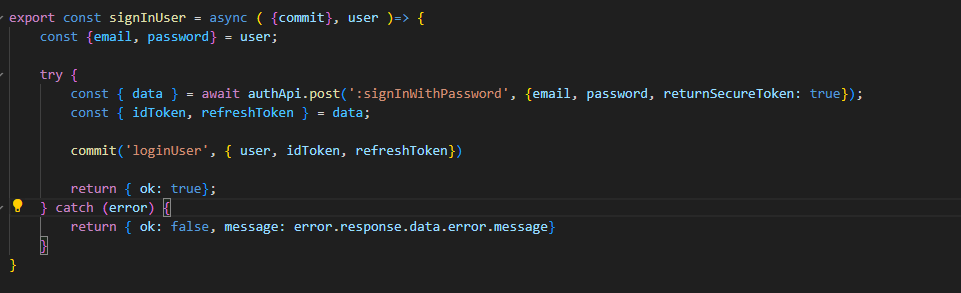
import { useRouter } from 'vue-router';





**LogIn de usuario:**

En el actions del store del auth hacemos:



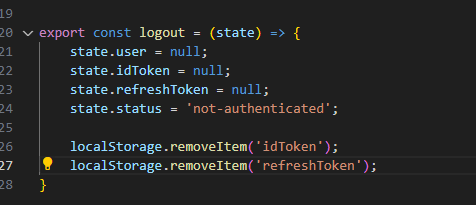
En el composable agregamos el loginUser



Hacemos la lógica en el loginUser.vue   


**Comprobar el estado del idToken:**

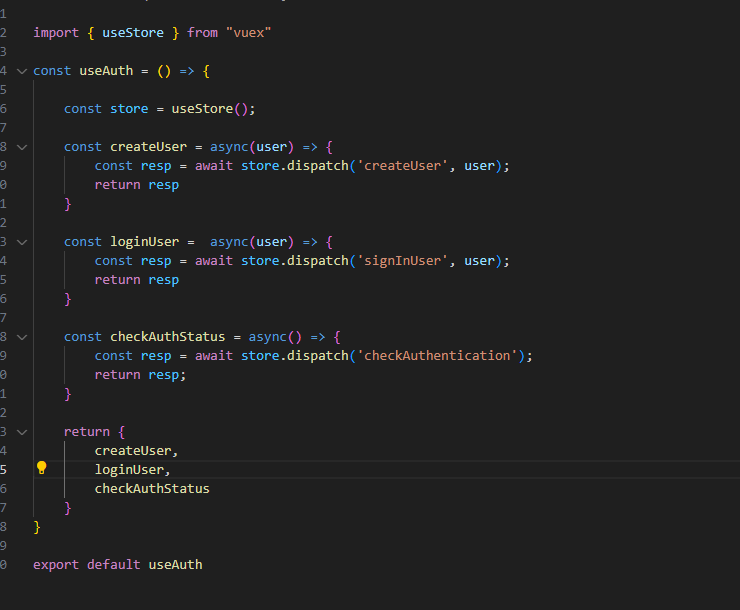
Creamos esto en las mutaciones del auth



Y creamos la acción



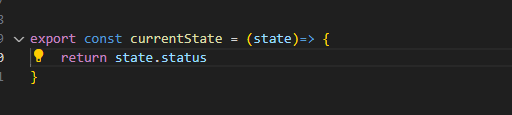
También agregamos esto al composable

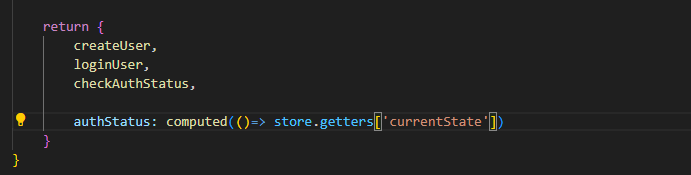


**Recargar el estado de la autenticación:**

Lo hacemos en el app.vue que es el router principal

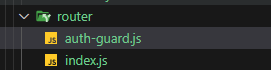
Vamos a crear un getter

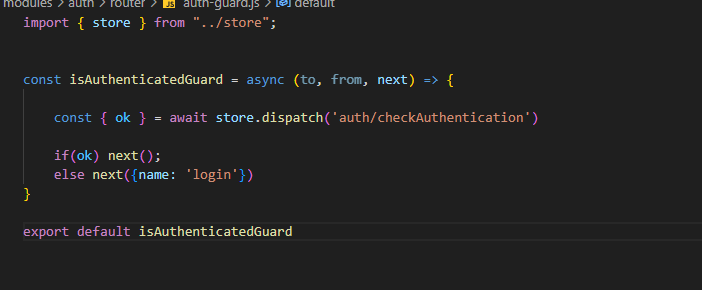


Y en el composable, es decir el useAuth vamos a retornar eso   


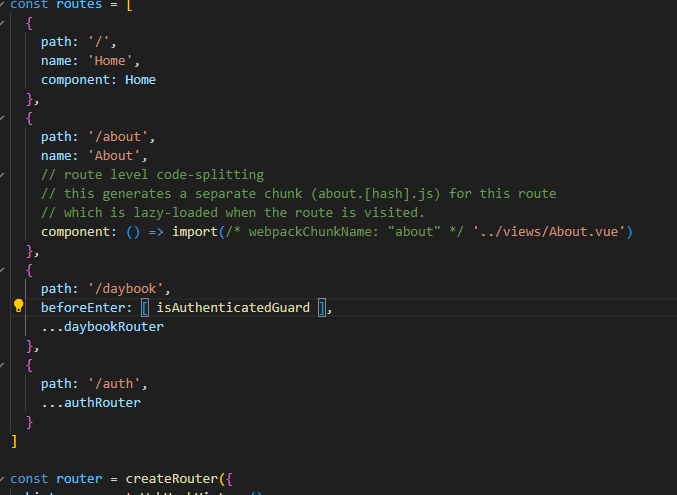
**AuthGuard, protección de rutas:**

Creamos un guard en la carpeta router



Ahí verificamos si está todo ok vamos a redirigirlo a la pagina que quiera ir, sino vamos a redirigir al login, saca la info del store, el composable no es accesible xq no esta en un ciclo de vida de vue esto  


Y lo llamamos del router con beforeEnter

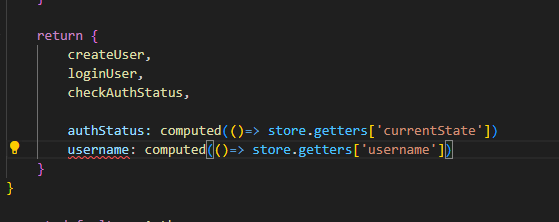


**Logout y nombre de usuario conectado:**

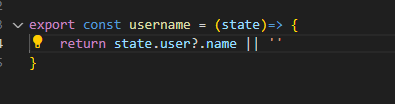
Esto en el componente del logout



En el useAuth



Getter



En el useAuth también hacemos el logout

