En esta versión se puede

1ro- a un html agregarle simplemente la librería js de vue

2do- declarar un div id app

3ro- crear un archivo a parte js donde se va ha realizar la unión

* Const app=new Vue({

el:’#app’

,data:{variable1:’valor1’,variable2:’valor2’}

,methods:{

… para eventos y demás

Usar this. Para acceder a la variables de data

}

,computed:{

… métodos computados

}

,beforeCreate (){

Se ejecuta al principio del ciclo de vida

}

,created (){

Al crear los métodos, observadores y eventos, pero aun no accede al DOM, aun no se puede acceder como tal al elemento HTML

}

,beforeMount (){

Se ejecuta antes de insertar el DOM

}

,mounted (){

Se ejecuta al insertar el DOM

}

,beforeUpdated (){

Al detectar el cambio

}

,updated (){

Al realizar los cambios

}

,destroyed (){

Se destruye toda la instancia

}

})

4to- usar las variables en la plantilla div app {{ variable1 }}

En el v-for usar a of para listas,

v-model=”variable2”

sele agrega como atributo a un input y se le pasa como valor una variable (de app) donde se valla aguardar el valor

:class=”variable” agrega clases, lo que pide es string o una lista de string , o un objeto donde sus ke yagan referencia la nombre de variables string y sus valores sean boleanos donde solo se aplicara la clase css si es verdadero

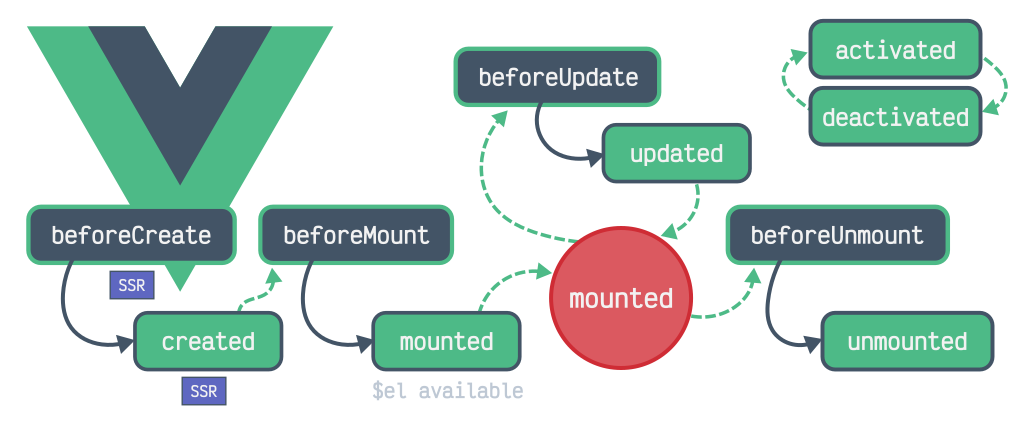
se le puede pasar explisitamente

:class=”{

‘nombre\_variable’:variable\_predicate, …

}”

Los objetos en el :class se pueden combinar con los computed y que estos computed retornen un objeto del tipo anterior



Para Crear componentes

Vue.component(‘nombre\_componente’,{

template: ’codigo html…

{{ variable1 }} o {{ atributo-html1 }}

’

,props: [‘atributo-html1’, … ] //para cuando se llame

//< nombre\_componente atributo-html1=”valor”>

// o < nombre\_componente :atributo-html1=”variable”>

// la diferencia es que sin los : seria literal

,data (){ //para retornar las variables a usar en este componente

return {

variable1: valor , variable2: valor2

}

}

,métodos de ciclo de vida

});

Luego dentro de <div id=” app” > < nombre\_componente> … < nombre\_componente> </div>

Una manera de mandar un dato de un componente hijo a un padre es

-1ro en el hijo dentro de alguna función que se llame en el algún momento

this.$emit(‘atributo\_hijo\_evento’, variable\_valor );

se llama a esta función que recibe el nombre que va tener uno de los atributos de la etiqueta hijo

-2do en donde se use al hijo, seguro dentro del padre, en su etiqueta agregarle como atributo al @atributo\_hijo\_evento, recordar que el @ lo marca como evento,

Como en todos los eventos la variable $event representa el valor pasado a este evento, que en este caso es ‘variable\_valor’,

Se usa el @atributo\_hijo\_evento=’ código que usa a $event’

<hijo @atributo\_hijo\_evento=’ código que usa a $event’ …. > …</>

Ejmpl: <hijo @atributo\_hijo\_evento=’ variable\_de\_padre=$event’ …. > …</>



<https://vuex.vuejs.org/installation.html#direct-download-cdn>

para variables globales usar a vuex

const store = new Vuex.Store({

state : {

variable\_global: valor

},

mutations: {

método\_modifcar\_global () {

this.state. variable\_global = nuevo\_valor

}

, método\_modifcar\_global2 (state) {

state. variable\_global = nuevo\_valor

}

}

actions:{ //son métodos asincornos, que reciben como parámetro un { commit }

// desde donde se pueden llamar a las mutaciones con el

// commit(‘nombre\_mutacion’)

//es buena idea usar aquí a fetch

metodo\_accion: async funtion( { commit } ){

commit(‘nombre\_mutacion’)

}

}

})

new Vue({

….

, store: store

})

Para llamar a las variables globales

en algún template <…> {{ store.state. variable\_global }} </..>

agregar en el computed a Vuex.mapState([‘variable\_global1’, …….. ]) con las variables que se van a usar en este componente

Para llamar a lo métodos mutations

En alguna parte del código ejecutable

$ store.commit(‘método\_modifcar\_global’)

Para usar variables globales en componentes

Se puede crear un método de computed que retorne un store.state. variable\_global

Vue.component(‘nombre\_componente’,{

…

,computed{

….

, método\_variable(){ return store.state. variable\_global ;}

}

})

O se puede usar al método Vuex.mapState([‘variable\_global1’, …….. ]) que retorna un objeto con las variables que le pidan

Vue.component(‘nombre\_componente’,{

…

,computed{

….

,… Vuex.mapState([‘variable\_global1’, …….. ]) //los 3… son por ema6

}

})

Esto ultimo permite al componente usar la variable\_global en el js y en el template

Para usar los métodos mutations del store, ósea de las variables globales, se mapean igual que con las variables, pero se mapean dentro de methods

Y lo mismo con las acciones

Vue.component(‘nombre\_componente’,{

…

, methods {

….

,… Vuex.mapMutations ([‘método\_modifcar\_global’, …….. ]) //los 3… son por ema6

,… Vuex.mapActions ([‘metodo\_accion1’, …….. ])

}

})