# Crear proyecto

* vue create NOMBRE\_PROYECTO

vite + js + pnpm

* pnpm create vite@latest nombre-del-proyecto --template vue

vite + pnpm (y luego seleccionar vue para decidir si js o ts)

* pnpm create vite@latest nombre-del-proyecto

# Estructura básica de un proyecto

## El archivo CP/vue.config.js

Para las diferentes configuraciones sobre posibles pluguins

## El archivo CP/tsconfig.json

Solo aparece si se seleccionó TypeScript, almacena las configuraciones de sobre este idioma

## El archivo CP/package.json

Archivos de configuración y sus dependencias

## El archivo CP/package-lock.json

Archivos de configuración y sus dependencias, descripción más exhaustiva de esto

## El archivo CP/ .eslintrc.js

Para las diferentes configuraciones sobre el Linter?

Tiene que ver con los tipos de entornos

## El archivo CP/ .browserslistrc

Descripción de los navegadores a los que les damos soporte

## CP/public/

Aquí se ubican los archivos públicos de nuestro proyecto, incluyendo el punto de entrada, index.html

## El archivo CP/ public /index.html

Donde se encuentra la raíz de ejecución del proyecto, porque toda la aplicación va a ser sustituida por la etiqueta <div id="app"></div>

## CP/src/

Carpeta de trabajo, aquí desarrollaremos nuestros componentes vue

## CP/src/shims-vue.d.ts

Declaración de tipos y módulos necesarios para TypeScript

## CP/src/main.ts

Archivo encargado de montar el index, el componente principal de la aplicación (App.vue)

## El archivo CP/src/App.vue

En el se escribe código html, ósea también se puede agregar el <scrips>

## CP/src/components/

Donde se almacenan los componentes .vue que creemos

## CP/assets/

Donde se encuentran lo elementos que los componentes pueden llegar a necesitar, como las imágenes

La norma para nombrar los componentes son dos palabras, cada una en Mayúsculas y unidas

# Sintaxis de plantillas

* Utiliza una sintaxis de plantillas basada en html que le permite vincular declarativamente el dom
* Todas las plantillas de VUE son html sintácticamente validas
* Por debajo compila las plantillas en un código JS altamente optimizado
* Combinando con la reactividad , es capas de calcular la misma cantidad de componentes a renderizar y aplicar la cantidad mínima de manipulaciones del DOM

Interpolación de textos: con llaves dobles es como se utilizan las variables dentro del código html (tipo django) (bigotes)

<script setup> o <script lang="ts">

const variable ="uno";

</script>

El script es el que va almacenar la logia del componente

<template>

  <h1>hola {{ variable }}</h1>

<OtroComponente …>…</OtroComponente>

</template>

Se puede usar un template dentro de otro elemento html que también este dentro un template, (como no es una etiqueta html solo se renderiza su contenido )

Enlaces a atributos (v-bind):

Los bigotes (llaves dobles) no se pueden insertar dentro de los atributos html, En su lugar se utilizar v-bind para enlazar el valor de un atributo

Delante del atributo se pone v-bind:nombre-atributo=”variable” y no es necesario usar las llaves dobles

<h1 v-bind:style="variable">

También se puede solo poner los : sin la palabra v-bind

<h1 :style="variable">

Los bigotes aceptan expresiones JS y por lo tanto también el contenido de los v-bind

<h1 :style="variable.toUpperCase()">

<h1>hola {{ variable.toUpperCase() }}</h1>

<h1>hola {{ ` texto ${variableArreglo[2]}` }}</h1>

Las variables dentro de los bigotes son interpretadas como su forma string por lo que si se le pasa un [] ser vera como su ”[a,b,c]”

Los bigotes no aceptan condicionales, pero si acepta ?:

<h1>hola {{ variable?”fue v”:”fue f” }}</h1>

# :class

:class=”variable” agrega clases, lo que pide es string o una lista de string , o un objeto donde sus key hagan 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

# :style

:style=”{

‘propiedad’: variable // en kebab-case entre ‘ ’

O , nombrePropiedad: variable //en camerCase

}”

# Directivas

* Son atributos especiales con el prefijo v-
* El trabajo de una directiva es aplicar actualizaciones de forma reactiva al DOM
* (mio) Son como atributos especiales del html
* v-if
  + se usa para mostrar o ocultar un elemento de la plantilla
  + <h2 v-if="mostrar">texto a mostrar1</h2>
* V-else
  + Se tienen que poner en la siguiente etiqueta al v-if
  + <h2 v-if="mostrar">texto a mostrar1</h2>
  + <h2 v-else>texto a mostrar2</h2>
* V-else-if
  + Se tienen que poner en la siguiente etiqueta al v-if
  + <h2 v-if="mostrar">texto a mostrar1</h2>
  + <h2 v-else-if="mostrar2">texto a mostrar2</h2>
  + <h2 v-else-if="mostrar3">texto a mostrar3</h2>
  + <h2 v-else>texto a mostrar4</h2>
* V-show
  + Para mostrar o ocultar un elemento de la plantilla
  + Solo cambia la propiedad display del elemento
  + V-if tienen costos de alternancia mas altos mientras que v-show tiene costos de renderización inicial mas alto
  + <h2 v-show="mostrar">texto a mostrar1</h2>
* V-for
  + Para iterar
  + Se puede darle un atributo key a cada elemento si se quiere reordenar
* <p v-for="(elemento,indice) in lista" :key=”indice”>{{ elemento }} {{ indice }}</p>
  + Repite la etiqueta en la que esta incluido
  + Cuando existen en una misma etiqueta un v-if y un v-for , el v-if tienen una prioridad mas alta por lo que la condición del v-if no puede acceder a las variables del v-for

# Eventos

* v-on
  + normalmente se acorta con el símbolo @ para escuchar eventos DOM y ejecutar JS cuando se activen
  + v-on:click=”metodo” o @click=”metodo”
* modificadores
  + se le agregan a un evento para especificar aún más cuando es que se va activar
  + se pueden encadenar
  + evento.modificador1.modificador2
  + todos
    - prevent
      * quita el efecto por defecto
  + click
    - left
    - right
    - middle
  + keyup
    - enter
    - el key code ejem: keyup.12

# Reactividad (Ref)

* Ref()
  + Es una referencia reactiva
  + Toma un argumento y lo devuelve envuelto dentro de un objeto con una value propiedad, que luego puede usarse para acceder o mutar el valor de una variable reactiva
  + DOM cuando muta el estado reactivo, el DOM se actualiza automáticamente
  + import {ref,Ref} from 'vue';
  + const vractiva:Ref=ref(0);
  + vractiva.value++;
  + <p>{{ vractiva }}</p>
  + Recibe un valor por defecto
  + Luego para acceder y manipular su valor es con vreactiva.valor
  + En el template no es necesario acceder al valor con .value
* Computed()
  + Sirven para generar cálculos en nuestros componentes
  + Es una función que retorna un valor pero que en la plantilla se va a tratar como una variable (como es una función se adapta a las circunstancias del momento)
  + import {computed} from 'vue';
  + const vcomputada= computed ( ()=>{return vractiva.value+1;} )
  + <p>{{ vcomputada }}</p>
  + Al contrario que los métodos, esta se almacenan en cache
  + Solo se volverá a evaluar cuando una de sus propiedades reactivas haya cambiado

Para compilar el código es

CP> npm build

CP>yarn build

Crea un carpeta dist

* + - * + yarn add sass-loader sass
        + yarn add less-loader less

# style scoped lang

<style scoped lang=”tipo ejmpl scss”>

…

</style>

El style del componente, para decirle que estos estilos solo aplican para este componente usar a scoped

# componentes uno dentro de otro

Como los componentes suelen estar en la misma carpeta para utilizar uno dentro de otro, sobre todo en el script, primero hay que importarlo

Import NombreComponente from ‘./NombreComponente.vue’

El ‘.’ En la dirección quiere decir que es en esta misma carpeta

Luego se puede usar en el <template> con <NombreComponente> …</>

# OptionApi

De la forma OptionApi (lo que se usaba en el 2)

Si queremos definir nuestro propio componente hay que hacer uso del defineComponent de VUE

Import { defineComponent } from “vue”

Ahora se define el componente

Export deafult defineComponent({

name:”NombreEtiquetaComponente”

,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

O {

nombreAtributoHtml:{

type: ejmpl String

,required: variable boleana

,default: variable por defecto

}

, …

}

{Nombre:Nombre}

,data(){

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

}

,methods:{

Metodo(){…}

… para eventos y demás

Usar this. Para acceder a la variables de data

}

,watch:{

// se le pasan como valor a un v-model=”metodoObservador”

metodoObservador(value,oldValue){…}

}

})

De la forma CompositionApi

## defineComponent

Si queremos definir nuestro propio componente hay que hacer uso del defineComponent de VUE

Import { defineComponent } from “vue”

Ahora se define el componente

Export deafult defineComponent({

name:”NombreEtiquetaComponente”

,props: … //ver el anterior

,emits:[“nombre\_atributo\_evento”,] // evento declarado como en esta etiqueta cuando

// sea un hijo < NombreEtiquetaComponente @nombre\_atributo\_evento=”evento”

,setup(){ o (props, { emit } ){

No hay que usar el this porque todo esta ocurriendo dentro de la función

//para llamar a los eventos pasados como atributos @nombre\_atributo\_evento

// usar emit(atributo, los argumentos que necesite )

Return {variable1,variable2}

}

})

# setup script

Si se incluye el setup en el script ya no es necesario el defineComponent y se puede trabajar directamente como si estuviéramos dentro del setup(){}

Automáticamente exporta los componentes para que el template los tenga accesibles (no es necesario un return al final )

# v-model

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

# consumir una API (CP/src/services/)

Si se va a consumir una API, lo mas correcto es poner el código aparte, generalmente en una carpeta CP/src/services/

Y hay NombreServicio.js o .ts, según el lenguaje

class NombreServicio{

    private atributo:Ref<Array<IAtributo>>

    constructor(){ this.atributo=ref([]);} // pq generalmente es una lista de cosas

    getAtributo():Ref<Array<IAtributo>>{ return this.atributo }

    async fetchAll():Promise<void>{

    //obtener el el dato mediante fetch y darselo al atributo

    }

    async fetchById(id:string|Array<string>):Promise<void>{

        //obtener el el dato mediante fetch y darselo al atributo

    }

}

Export default NombreServicio;

# “@/” “src/”

A la importar algo interno de nuestro proyecto, en la ruta se puede usar “@/” en ves de “src/”

# ciclo de vida

Para llamar a lo métodos del ciclo de vida, se pueden usar hubs ejmpl;

Import { onMounted } from ‘vue’

onMounted( async ()=>{

//por ejemplo llamar a los servicios

Await variableNombreServicio.fetchAll()

} )

# CP/src/scss

Cuando se trabaja con scss se suele crear una carpeta CP/src/scss

Donde se ubiquen los datos relacionados al mismo

Ejmpl: \_variables.scss

Luego modificar el vue.config.js

module.exports = defineConfig({

  transpileDependencies: true //esto esta por defecto

  ,css:{ //es esto lo que se agrega

    loaderOptions:{

      sass:{

        additionalData:`

          @import "@/scss/\_variables.scss"

        `

      }

    }

  }

})

# bootstrap

Hay que recargar el servicio porque puede que no lo detecte de forma automática

* + - * + yarn add bootstrap
        + yarn add @popperjs/core

para usarlo agregarle al main.ts

import "bootstrap/dist/css/bootstrap.min.css"

import "bootstrap"

# Router

* + - * + yarn add vue-router

Cuando se usa Router, se queda modificado el main.ts

Import router form ‘./router’

createApp(App).use(router).mount(‘#app’)

además se Agregan dos carpetas CP/src/router y views

dentro de CP/src/router esta index.ts

const routes: Array<RouteRecordRaw> = [

  {

    path: '/', //url de la ruta

    ,name: 'NombreRuta', //nombre de la ruta

    ,component: HomeView //vista de la ruta

,meta:{

Atributo\_extra\_1: valor

…

}

  },

{

    path: '/ruta', //url de la ruta

   redirect: ‘url path’ //vista de la ruta

  },

{

    path: '/ruta', //url de la ruta

   redirect: v=>{ return {name:‘NombreRuta’}} //vista de la ruta

  }

,

  {

// usar esta manera para una carga lazy del

// contenido, es mas aconsejable siempre usar esta

// forma

    path: '/about',

    name: 'about',

    // es importante que se ponga el comentario

// dentro del import, pq es el nombre del archivo

// que indica que se va a crear un JS a parte para

// que se carguen los datos de forma separada de esa

//vista

    component: () => import(/\* webpackChunkName: "nombreDelJSaCrear" \*/ '../views/NombreVista.vue')

  }

, …

]

const router = createRouter({

  history: createWebHistory(process.env.BASE\_URL),

  routes

})

router.beforeEach((to,from,next)=>{

  const estaAutorizado=false;

  const necesitaAutorizacion=to.meta.parametro\_necesita\_autorizacion;

  if(!estaAutorizado && !necesitaAutorizacion){

    next('nombre\_ruta\_login');

  }

  next()

})

## CP/src/views

En este archivo se encuentra los datos sobre las diferentes rutas

Lo que hay en CP/src/views son componentes de vista

### <router-view>

Ahora el App.vue en su template va a usar un <router-view>

<template>

  <nav>

    <router-link to="/">Home</router-link> |

    <router-link to="/about">About</router-link>

<router-link to="{name:‘NombreRuta’}">Algo

</router-link>

<router-link :to="{name:‘NombreRuta’

,params:{parametro1:valor1

, parametro2:valor2 ,…}

}">Algo

</router-link>

  </nav>

  <router-view/>

</template>

O si el App.vue no contiene datos para la vista ya que de las vista se van a encargar los componentes y tal ves hay un componente que representa el home, el template del App.vue quedaría

<template>

    <router-view/>

</template>

### <router-link>

Para redirigir a una nueva ruta, poner en cualquier template

<router-link :to="{name: nombreRuta }">Ruta</router-link>

<router-link :to="{

name: nombreRuta

params: {

parametro1: valorPorDefecto

}

}

">Ruta</router-link>

Para definir rutas con parámetros, modificar el index.ts

const routes: Array<RouteRecordRaw> = [

  {

    path: '/ruta/:parametro1/:parametro2',

    name: 'nombreRuta',

    component: VistaView

  }, …

]

## rutas dentro de un componente

Para hacer uso de los datos de las rutas dentro de un componente

Import {useRouter} from ‘vue-router’

Generalmente se usa luego dentro del onMounted

const route = useRoute();

const parámetro=route.params.nombreParamatro;

# FireBase

https://console.firebase.google.com/project/curso-vue-d391a/authentication/users?hl=es

Para usar FireBase (sistema de bakend automático para autentificaciones provisto por google)

* + - * + yarn add firebase

en el main.ts

// firebase configuracion

// Import the functions you need from the SDKs you need

import { initializeApp } from "firebase/app";

import { getAnalytics } from "firebase/analytics";

// TODO: Add SDKs for Firebase products that you want to use

// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration

// For Firebase JS SDK v7.20.0 and later, measurementId is optional

const firebaseConfig = {

  apiKey: "AIzaSyA5SjxPxkG\_OLLpg8H4--EYsLenXNejQ-8",

  authDomain: "curso-vue-d391a.firebaseapp.com",

  projectId: "curso-vue-d391a",

  storageBucket: "curso-vue-d391a.appspot.com",

  messagingSenderId: "888487065004",

  appId: "1:888487065004:web:3f392fa635c6ef3b45b609",

  measurementId: "G-T26G62QC94"

};

// Initialize Firebase

const app = initializeApp(firebaseConfig);

const analytics = getAnalytics(app);

// end firebase configuracion

Antes de

createApp(App).use(router).mount('#app')

Luego en un componente tal ves Loguin

import {getAuth, signInWithEmailAndPassword} from 'firebase/auth';

let email=ref("");

let password=ref("");

const authUser=()=>{

    const auth=getAuth();

    signInWithEmailAndPassword(auth,email.value,password.value)

    .then(()=>{

        alert("se autentico!!");

    })

    .catch((error)=>{

        alert("dio error "+error.message);

    });

}

Si es a través de Google

import {GoogleAuthProvider,signInWithPopup,getAuth} from 'firebase/auth'

const googleProvider=new GoogleAuthProvider();

const auth=getAuth();

const loginGooogle=()=>{

    signInWithPopup(auth,googleProvider)

    .then((result)=>{

        const credential=GoogleAuthProvider.credentialFromResult(result);

        const token=credential?.accessToken;

        alert('login success');

    })

    .catch((errors)=>{

        alert("no lo logro");

    })

};

# VUEX

Al usar VUEX, hay que modificar el main.ts

import { createApp } from 'vue'

import App from './App.vue'

import router from './router'

import store from './store'

createApp(App).use(store).use(router).mount('#app')

## CP/src/store/index.ts

Y se crea el archivo CP/src/store/index.ts

import { createStore } from 'vuex'

export default createStore({

  state: {

variable\_global: valor

  },

  getters: {

//métodos geters no iguales a la variables state

// son como computeds

Obtener\_variable\_modificada(state){

Return state.variable;

}

  },

  mutations: {

//métodos para modificar los state, son sincronas

método\_modifcar\_global () {

this.state. variable\_global = nuevo\_valor

}

, método\_modifcar\_global2 (state) {

state. variable\_global = nuevo\_valor

}

, método\_modifcar\_global3 (state, nuevo\_valor) {

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’)

}

}

  },

  modules: {

//para subdividir el store

  }

})

Dentro del código de algún componente, para llamar a alguna variable directamente del store

V=this.$store.state.counter

Para llamar a los geters, no se llama con ()

V=this.$store.getters.nombre\_metodo\_geter

Para llamar a las mutaciones, es a través del commit, pasándole el nombre del método y su argumento

this.$store.commit(‘método\_modifcar\_global3’,valor nuevo)

Tambien se pueden usar las variables y métodos del store, mapeando todo su contenido

import { mapMutations, mapState, mapGetters, mapActions} from 'vuex'

dentro de la definición del componente

,computed{

….

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

}

, methods {

….

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

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

}

, methods {

….

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

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

}

## useStore

Otra forma de usar el store es con useStore

import { computed} from 'vue'

import { useStore} from 'vuex'

const store=useStore()

const variable=computed(()=>store.state.variable\_global)

const variable\_de\_geter=computed(()=>store.getters.obtener\_variable)

const metodo\_mutacion=()=>store.commit('método\_modifcar\_global3', variable +' otro valor')

# pinia

Para trabajar con pinia

* + - * + yarn add pinia
        + pnpm install pinia

modificar el main.ts

import { createPinia } from 'pinia'

const pinia=createPinia()

createApp(App).use(pinia).use(router).mount('#app')

crear el archivo CP/src/store/nombre\_modulo.ts

import { defineStore } from 'pinia'

export const useContadorStore=defineStore('contador',{

    state:()=>{

        return {

            contador:0

        }

    },

    getters:{

        obtener\_estado:(state)=>state.contador

    },

    actions:{

// pueden ser síncronos y asíncronos

        incrementar(valor=1){

            this.contador+=valor;

        }

    }

})

O de un modo un poco mas simple, donde se define una funcion que retorna la variable y el seter

import { defineStore } from "pinia";

import {ref} from 'vue';

export const useMiStore=defineStore('weather',()=>{

    const nombrepersona=ref("ninguno");

    const setNombrePersona=(nombre:string)=>{

        setNombrePersona.value=nombre;

    }

    return {nombrepersona,setNombrePersona}

})

Luego el composable

Src/composables/usePersona.ts

import { useMiStore } from "../stores/miStore";

import { getPersona } from "../helpers/getPersona";

export const usePersona=async ()=>{

    const persona=await getPersona(1);// un helper que llama a la api

    const personaStore=useMiStore();

    personaStore.nombrepersona=persona.nombre;

    }

Su uso en la plantilla

<template>

    <h1>El nombre de la persona es: {{personaStore.nombrepersona}}</h1>

</template>

<script setup lang="ts">

    import { usePersona } from '../composables/usePersona';

    import { onMounted } from 'vue';

    import { useMiStore } from '../stores/miStore';

    const personaStore=useMiStore();

    onMounted(async ()=>{

        usePersona();

    })

</script>

# cookies

Para trabajar con la cookies

* + - * + yarn add vue-cookies

Primero hay que modificar el main.ts

import { VueCookies } from 'vue-cookies'

createApp(App),use(VueCookies {

  expires:'1d' // this sets a path attribute on each cookie which allows us to share cookies across subdomains using vue-

  ,path:'/' //No necesario

  ,domain:'' //No necesario

  ,secure:'' //No necesario

  ,sameSite:'' //No necesario

}

).use(pinia).use(router).mount('#app')

Luego

$cookies.set('nombre\_cookie',1000)

    $cookies.get('nombre\_cookie')

    $cookies.remove('nombre\_cookie')

    $cookies.isKey('nombre\_cookie')

    $cookies.keys()

# sesiones

Para usar las sesiones

* + - * + yarn add vue-session

modificar el maint.ts

import VueSession from 'vue-session'

createApp(App).use(pinia).use(router).use(VueCookies,{expires:'1d' }).use(VueSession).mount('#app')

luego usar con

$session.start()

    $cookies.set('nombre\_key\_session',1000)

    $session.get('nombre\_key\_session')

    $session.id()

    $session.renew()

    $session.destroy()

    $session.remove('nombre\_key\_session')//no se

    $session.isKey('nombre\_key\_session')//no se

    $session.keys()//no se

# Layout

Para tener un componente que haga de plantilla, para que donde se utilice solo se modifiqué una parte, se le agrega en algún lado un solo <slot></> vacío por dentro

Estos componentes es buena idea almacenarlos en una carpeta CP/src/layouts/

<ComponenteLayout> … <slot></> … </>

Y luego donde se quiera reutilizar este componente, se pone y lo que se escriba dentro de del sustituirá automáticamente la parte del slot, donde se utilice hay que importar con js el ComponenteLayout

<ComponenteLayout> código distinto </>

Si van a existir varios slot en un mismo Layout, a cada uno se le pone un atributo name=“nombreDeSlot” y lego se definen dentro de <templates v-slot: nombreDeSlot o # nombreDeSlot> código nuevo</>

<ComponenteLayout> …

<slot name=“nombreDeSlot1” ></>

<slot name=“nombreDeSlot2” ></>

… </>

Lugo donde se use

<ComponenteLayout>

<templates v-slot:nombreDeSlot1> código nuevo</>

<templates #nombreDeSlot2> código nuevo</>

</>

# composable

Para reutilizar conjuntos de variables, o se podría decir funciones que retornen variables, se utiliza un estructura de código llamada composable, que consiste en dentro de la carpeta

La nomenclatura de su nombre debe de ser useNombreDeElemento, osea que siempre comience con “use”

CP/src/composables/nombreFuncionDeComposable.ts

//Crear una función

Export funtion nombreFuncionDeComposable(){

Código

Return { variable1,variable2, ….}

}

Luego dentro de algún componente

Importar el nombreFuncionDeComposable

Const { variable1,variable2} = nombreFuncionDeComposable();

Suele ser usado como apoyo a pinia y llamadas a la api, donde dentro del composable se llama al “helper” y se actualiza el “store”

import { useMiStore } from "../stores/miStore";

import { getPersona } from "../helpers/getPersona";

export const usePersona=async ()=>{

    const persona=await getPersona(1);// un helper que llama a la api

    const personaStore=useMiStore();

    personaStore.nombrepersona=persona.nombre;

    }

# cargar de forma dinámica un componente (defineAsyncComponent)

Para cargar de forma dinámica un componente y a su ves poder cambiar este componente por otro de forma dinámica

-1ro: cargamos el componente usando a

import {ref,defineAsyncComponent} from 'vue';

const componente1=defineAsyncComponent(()=>import('@/components/Componente1.vue'))

const componente2=defineAsyncComponent(()=>import('@/components/Componente2.vue'))

2do: creamos una variable que va a almacenar el componente en uso actual

const componenteAUsar=ref(componente1);

3ro: creamos un método que se va a encargar de cambiar el componenteAUsar

const ponerComponente=(cmp)=>componenteAUsar.value=cmp;

4to: de ser necesario creamos los datos que se van a pasar como argumentos

const datosAPasar=ref([

  {nombre:'Juan',edad:30},

  {nombre:'Pedro',edad:45},

  ]);

-5to: en la plantilla definimos el sitio donde va a ir nuestro componente dinámico, para ello usamos a

<component :is="componenteAUsar" :atributo-del-componente="datosAPasar" ></component>

Aquí :is es la variable que almacena el componente que va a variar

-6to: en algún lado se usa el método que cambia el componente, por ejemplo en un botón

<button @click="ponerComponente(componente2)">texto</button>

-7mo: Luego dentro del componente para acceder a los datos que se le pasaron

import {defineProps} from 'vue';

const propiedades=defineProps(['atributo-del-componente1'])

# transiciones



## <Transition>

Para aplicarle transiciones a un componente hay que encerrarlo dentro de un

<Transition> Componente o Contenido a animar </>

Y luego poner las clases css

.v-enter-active, .v-leave-active, animación de entrada

.v-enter-from ,.v-leave-to animación de salida

Usar el v-if sobre el contenido a aplicar

<Transition> <componente\_o\_contenido\_a\_animar v-if=”variable”>…</> </>

## <TransitionGroup>

Para aplicarle las misma transiciones a un conjunto de componentes, tienen que estar dentro de un <TransitionGroup name=”prefijocss” tag=”nombre de la etiqueta que este va sustituir” >

Por ejemplo si el conjunto de elementos donde todos serian <li>, entonces le aria falta un <ul>, pero si se encierra al <ul> entonces solo se le aplicaría al <ul>, por eso es necesario sustituir al <ul> por < TransitionGroup> y decirle mediante su atributo tag=”ul” que la etiqueta que esta sustituyendo es un <ul>

. prefijocss-enter-active, . prefijocss-leave-active, animación de entrada

. prefijocss-enter-from ,. prefijocss-leave-to animación de salida

Como se puede ver si hay varios grupos, se pueden distinguir sus transiciones por el prefijocss que se ponga

# los iconos de vue

Si se quieren usar los iconos de vue

* + - * + yarn add oh-vue-icons

<https://oh-vue-icons.js.org/>

https://oh-vue-icons.js.org/docs

Hay que modificar el main.ts

import { OhVueIcon, addIcons } from "oh-vue-icons";

import { Icono1,Icono2 } from "oh-vue-icons/icons";

app2.component("v-icon", OhVueIcon);

Luego se utilizan en las plantillas con y el nombre del icono se pone esta ves con kebab-case, osea cambiar todas las mayúsculas por minúsculas y separar las palabras con –

 <v-icon name="icono-2" />

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Accepted Values** | **Default value** |
| **scale** | Icon size | **number** | / | **1** |
| **animation** | Type of animation | **string** | **wrench** / **ring** / **pulse** / **spin** / **spin-pulse** / **flash** / **float** | / |
| **speed** | Animation speed | **string** | **slow** / **fast** | / |
| **hover** | Enable animation only when being hovered | **boolean** | **true** / **false** | **false** |
| **flip** | Used to flip icon | **string** | **vertical** / **horizontal** / **both** | / |
| **fill** | Fill color of icon | **string** | HEX color code or color name | **currentColor** |
| **label** | Icon lable | **string** | / | / |
| **title** | Icon title | **string** | / | / |
| **inverse** | Make icon color white? | **boolean** | **true** / **false** | **false** |

# Persistencia pinia

Para manejar persistencia de datos mediante pinia

https://github.com/prazdevs/pinia-plugin-persistedstate

* + - * + yarn add pinia-plugin-persistedstate

Modificar el main.ts

import piniaPluginPersistedstate from 'pinia-plugin-persistedstate'

const pinia=createPinia()

pinia.use(piniaPluginPersistedstate)

Luego en el store agregarle un atributo persist:true

import { defineStore } from 'pinia'

export const useContadorStore=defineStore('contador',{

    state…

    getters…

    actions…

    ,persist:true

})

# métodos observadores

Para usar los métodos observadores

## watchEffect

import { watchEffect } from 'vue'

watchEffect(()=>{

  // usar a una variable ref

});

## watch

O para una variable ref en específico

import { watch } from 'vue'

watch (variableRef,()=>{

  // usar a una variable ref

});

# funciones de un componente hijo en un padre defineExpose

Para usar funciones de un componente hijo en un padre, hay que referenciarlo para poderlo usar en el código

-1ro en el hijo hay que exponer los elementos

import { defineExpose } from 'vue';

defineExpose({metodoDelhijoExpuesto1,variableExpuesta2})

-2do en el padre usar al hijo

<Componentehijo ref="referenciaAComponente" />

-3ro

const referenciaAComponente=ref(null);

referenciaAComponente.metodoDelhijoExpuesto();

# acceder a todas los componentes que tengan el atributo ref .$refs

Se puede acceder a todas los componentes que tengan el atributo ref de un componente usando

this.$refs.referenciaAComponente

# carga lenta de imágenes

Para usar la carga lenta de imágenes

* + - * + yarn add vue3-lazyload
        + npm i vue3-lazyload

https://www.npmjs.com/package/vue3-lazyload

Modificar el main.ts

import VueLazyLoad from 'vue3-lazyload'

app.use(VueLazyLoad, {

,loading:'sample.jpg' //img mientras carga

      ,error:'sample.jpg' //img si da error

      ,delay:5000

lifecycle: { //si se incluye sustituye el orignal

loading: (el) => {

console.log('loading', el)

},

error: (el) => {

console.log('error', el)

},

loaded: (el) => {

console.log('loaded', el)

}

}

})

Luego en las etiquetas img

<img v-lazy="{

      src:'../assets/logo.png'

      ,loading:'sample.jpg' //img mientras carga

      ,error:'sample.jpg' //img si da error

      ,delay:5000

  }" >

Se pueden modifcar los estilos

<style>

img[lazy=loading] {

/\*your style here\*/

}

img[lazy=error] {

/\*your style here\*/

}

img[lazy=loaded] {

/\*your style here\*/

}

</style>

# GRAPHQL

Usar graphql

<https://graphql.org/>

* + - * + yarn add graphql graphql-tag @apollo/client
        + yarn add @vue/apollo-composable

## modificar el main.ts

### Para conectarse a una sola conexión

import { createApp,provide,h } from 'vue'

import { ApolloClient,createHttpLink,InMemoryCache } from '@apollo/client/core'

import { DefaultApolloClient } from '@vue/apollo-composable';

const httpLink=createHttpLink({

  uri:'http://localhost:4000/graphql'

})

const cache=new InMemoryCache()

const apolloClient=new ApolloClient(

  {link:httpLink,cache}

)

const app = createApp({

  setup(){

    provide(DefaultApolloClient,apolloClient)

  },

  render:()=>h(App),

});

### Para conectarse a varios servidores

import {ApolloClients } from '@vue/apollo-composable';

const app2 = createApp({

  setup(){

    provide(ApolloClients,{

      default:apolloClient1

      ,cliente1:apolloClient1

      ,cliente2:apolloClient2

    })

  },

  render:()=>h(App),

});

## Realizar query

Dentro de un componente

https://v4.apollo.vuejs.org/api/use-query.html

<button @click="refetch">Refrescar datos</button>

import { useQuery } from '@vue/apollo-composable';

import gql from 'graphql-tag';

const {result,loading,error, refetch }=useQuery(gql`

  query ($name){

    getAllComments{

      name

      text

    }

getCommentsFromUser(name:$name){

      name

      text

    }

  }

` ,()=>{

  name:"nombre1"// aquí los pares v valor

} ,{

  fetchPolicy:'network-only',

  pollInterval:5000

}

)

Donde result almacena la respuesta

Loading es un booleano de si se termino la consulta

Error los errores

El refetch es el evento de recargar los datos, debe enviarse a un onlick

### Con variables

const {result,loading,error, refetch }=useQuery(gql`

  query ($name){

   getCommentsFromUser(name:$name){

      name

      text

    }

  }

` ,()=>{

  name:"nombre1"// aquí los pares v valor

})

### Pasar opciones

,{

  fetchPolicy:'network-only',

  pollInterval:5000

}

#### fetchPolicy

* fetchPolicy: Customize cache behavior.
  + cache-first (default): return result from cache. Only fetch from network if cached result is not available.
  + cache-and-network: return result from cache first (if it exists), then return network result once it's available.
  + cache-only: return result from cache if available, fail otherwise.
  + network-only: return result from network, fail if network call doesn't succeed, save to cache.
  + no-cache: return result from network, fail if network call doesn't succeed, don't save to cache.

### Para refrescar los datos

-1ro

Tener en la desestructuración el refetch

-2do pasarla como evento a un onclick

<button @click="refetch">Refrescar datos</button>

## Mutaciones

<button @click="createComment({name:'valor1',text:'valor2'})">Llamar a mutacion</button>

import { useMutation } from '@vue/apollo-composable';

const {mutate:createComment ,loading:loandingComment,error:errorComment

,onDone,onError

}=useMutation(gql`

  mutation($name: String!, $text: String!){

    createComment(name:$name,text:$text){

      name

      text

    }

  }

`);

### Pasar las variables desde el boton

Se le pueden pasar las variables desde el evento del botón

@click="createComment({name:'valor1',text:'valor2'})">Llamar a mutacion</button>

### Pasar las variables desde la propia consulta

Igual que el anterior, al final de la consulta

const {mutate:createComment}=useMutation(gql`

  mutation($name: String!, $text: String!){

    createComment(name:$name,text:$text){

      name

      text

    }

  }

`,()=>{

  variables:{name:"nombre1"},

// aquí los pares v valor

});

### Actualizar el contenido

Luego de ejecutar una mutación es probable que sea necesario actualizar la información

Se agrega al mismo nivel de las variables

const {mutate:createComment}=useMutation(gql`

  mutation($name: String!, $text: String!){

    createComment(name:$name,text:$text){

      name

      text

    }

  }

`,()=>{

  variables:{name:"nombre1"},

update:(cache,{data:{createComment}})=>{

    let data=cache.readQuery({query:getAllComments});

    data={

      ...data, comments:[

        ...data.comments,

        {

          name:"algo",

          text:"otro"

        }//createComment

      ]

    }

    cache.writeQuery({query:getAllComments,data});

    return data;

  }

});

### onDone

Agregarlo al declarar la mutación en la desestructuración

onDone(()=>{

  //algo

})

### onError

Agregarlo al declarar la mutación en la desestructuración

onError(()=>{

  //algo

})

## Suscripciones

* + - * + yarn add subscriptions-transport-ws

### main.ts

Hay que modificar el main.ts

import { ApolloClient,createHttpLink,InMemoryCache,split } from '@apollo/client/core'

import {WebSocketLink} from '@apollo/client/link/ws'

import { getMainDefinition } from '@apollo/client/utilities'

const wsLink=new WebSocketLink({

  uri:`ws://localhost:4000/graphql`,

  options:{

    reconnect:true

  }

})

const link=split(

  ({query})=>{

    const definition=getMainDefinition(query);

    return (

      definition.kind==='OperationDefinition' &&

      definition.operation=== 'subscription'

      )

  },wsLink,httpLink

)

### En el componente

import {ref,defineAsyncComponent,watch} from 'vue';

import { useSubscription } from '@vue/apollo-composable';

const {result:resulSub

,loading:loadingSub

,error:errorSub

}=useSubscription(gql`

  subcription(){

    commentCreated{

      name

      text

    }

  }

`)

watch(

  resulSub

  ,data=>{

    //algo

  }

  )

# Pwa

* + - * + yarn global add http-server

si ya esta esta compilado (build)

* + - * + http-server dist/`

y ahora nos debe de salir el incono para instalar la aplicación

## manifest.json

en dist/manifest.json es donde están las características de nuestra aplicación

Es aconsejable copiarlo para public/

### display

El campo "display" dentro de "manifest.json" en una PWA (Progressive Web App) puede tomar los siguientes valores:

1. "fullscreen": La aplicación ocupa toda la pantalla y se ejecuta como una aplicación independiente sin la barra de direcciones del navegador.
2. "standalone": La aplicación se ejecuta como una aplicación independiente, pero puede mostrar la barra de direcciones del navegador.
3. "minimal-ui": La aplicación se ejecuta como una aplicación independiente, pero con una interfaz de usuario mínima, lo que permite que el contenido se muestre a pantalla completa si es necesario.
4. "browser": La aplicación se ejecuta en una pestaña del navegador como cualquier otro sitio web sin funcionalidades específicas de la PWA.

Estos valores indican cómo se debe mostrar y ejecutar la PWA en el navegador del usuario.

## vue.config.js

también se pueden declarar estas propiedades modificando el vue.config.js

const { defineConfig } = require('@vue/cli-service')

module.exports = defineConfig({

  transpileDependencies: true,

  pwa:{

    name:"nombre",

    themeColor:"#33ff44",//camel case en ves de Snake case

  }

})

## registerServiceWorker.ts

el src/ registerServiceWorker.ts es donde están los métodos que definen el comportamiento durante el estado en diferentes puntos del ciclo de vida de una pwa

# i18n

<https://vue-i18n.intlify.dev/>

* + - * + yarn add vue-i18n@9

## maint.ts

Modificar el maint.ts

import { createI18n } from 'vue-i18n'

import { messages } from './lang/messages'

const i18n=createI18n({

    locale: 'en',//localStorage.getItem('language') || navigator.language, // 这个地方可以从localstorage里面

    fallbackLocale:'en',

    messages

})

const app=createApp(App)

app.use(i18n)

## src/lang/messages.ts

crear src/lang/messages.ts con

export const messages={

    en:{

        messages:{

            hello:"Hello word",

            conVariable:"Hello {name}",

            conCantidad:"no products | one product | (cantidad) products",

            conCantidadSinVariable:"no products | one product |  products"

        }

    },

    es:{

        messages:{

            hello:"Hola mundo",

            conVariable:"Hola {name}",

            conCantidad:"sin productos | un producto | (cantidad) productos",

            conCantidadSinVariable:"sin productos | un producto | productos"

        }

    },

}

## Usar

En el componente

<h1>{{ $t("message.hello") }}</h1>

<h1>{{ $t("message.conVariable",{name:"valor"}) }}</h1>

<h1>{{ $t("message.conVariable",{name:"valor"}) }}</h1>

    <h1>{{ $t("message.conCantidad",{cantidad:0}) }}</h1>

    <h1>{{ $t("message.conCantidad",{cantidad:1}) }}</h1>

    <h1>{{ $t("message.conCantidad",{cantidad:9}) }}</h1>

<h1>{{ $t("message.conCantidadSinVariable",1) }}</h1>

## Seleccionar idioma

<select v-model="$i18n.locale">

      <option v-for="locale in $i18n.availableLocales" :key="`locale-${locale}`"

      :value="locale" >{{ locale }}</option>

    </select>

# Signal

* + - * + yarn add vue-signals

## Tipo Reac

Es para usar las variables reactivas como react, dividiendo el geters y seters

import { createSignal } from 'vue-signals/dist/solid';

const [variableReactiva,setVariableReactiva]=createSignal("por defecto")

### usar el seter

#### directamente el valor

Se puede usar pasando directamente el valor

setVariableReactiva(" - Modificado por función");

#### ver valor anterior

o viendo el valor anterior y retornar el nuevo

setVariableReactiva(prevState => prevState+" - Modificado por función");

#### plantilla

<p>{{ variableReactiva() }}</p>

Para el geter hay que usar los ()

## Tipo Angular

import { signal } from 'vue-signals';

const variableReactivaAngular=signal("valor por defecto")

### set

variableReactivaAngular.set(" - Modificado por función")

### update

variableReactivaAngular.update(prevState => prevState+" - Modificado por función")

### plantilla

<p>{{ variableReactivaAngular() }}</p>

# reactive

const vref=reactive({

  atributo:"algo"

})

igual que ref, pero mas orientado a los objetos

no permite que se reasigne su valor, solo modificaciones a sus atributos

# Crear plugins

Son para agregar funcionalidades o variables globales

## CP/plugins/nombrePlugin.ts

Poner en CP/plugins/nombrePlugin.ts

export default {

    install:(app,options)=>{

        app.config.globalProperties.$variableGlobal=()=>"valor o realizar accion";

   }

 }

## Main.ts

Luego hay que agregarlo en el main.ts

import example from '../plugins/example'

app.use(example,{

    variableGlobal:"valor1"

})

## Usar en template

<p>{{ $variableGlobal() }}</p>

# Cypress

<https://www.cypress.io/>

para pruebas totales de la aplicación

# Axios

* + - * + pnpm install axios

ejemplo llamada a una api sencilla sin protección

<script setup lang="ts">

    import axios from 'axios';

    import { onMounted } from 'vue'

    import {ref,Ref} from 'vue';

    const users:Ref=ref([]);

    onMounted( async ()=>{

        const res=await axios.get("http://127.0.0.1:5000/personas")

        users.value=res.data;

        console.log(users)

    } )

</script>

<template>

    <h1>Mi Componente</h1>

    <ul>

        <li v-for="(item,index) in users" :key="item.id">

            {{item.nombre}} - {{item.edad}}

        </li>

    </ul>

</template>

## Ejemplo: Helper+ store pinia + composable

Src/helpers/getPersona.ts

import axios from "axios"

const API\_URL="http://127.0.0.1:5000"

export const getPersona=async (id) =>{

    const res=await axios.get(`${API\_URL}/personas/${id}`)

    const persona=res.data;

    return persona;

}

Src/strores/miStore.ts

import { defineStore } from "pinia";

import {ref} from 'vue';

export const useMiStore=defineStore('weather',()=>{

    const nombrepersona=ref("ninguno");

    const setNombrePersona=(nombre:string)=>{

        setNombrePersona.value=nombre;

    }

    return {nombrepersona,setNombrePersona}

})

Luego el composable

Src/composables/usePersona.ts

import { useMiStore } from "../stores/miStore";

import { getPersona } from "../helpers/getPersona";

export const usePersona=async ()=>{

    const persona=await getPersona(1);// un helper que llama a la api

    const personaStore=useMiStore();

    personaStore.nombrepersona=persona.nombre;

    }

La plantilla

<template>

    <h1>El nombre de la persona es: {{personaStore.nombrepersona}}</h1>

</template>

<script setup lang="ts">

    import { usePersona } from '../composables/usePersona';

    import { onMounted } from 'vue';

    import { useMiStore } from '../stores/miStore';

    const personaStore=useMiStore();

    onMounted(async ()=>{

        usePersona();

    })

</script>

## Configuración por defecto

Por ejemplo para que siempre tenga una ruta base por defecto y talvez para que siempre acepte json

Src/services/Api.ts

import axios from "axios";

export default ()=>{

    return axios.create({

        baseURL:'http://127.0.0.1:5000',

        headers:{

            'Content-Type':'application/json',

            'Accept':'application/json',

        }

    })

}

Src/services/PersonasService.ts

import api from "./api";

export default{

    getPersonas(){

        return api().get('/personas');

    }

}

## Post

Api

import axios from "axios";

export default ()=>{

    return axios.create({

        baseURL:'http://127.0.0.1:5000',

        headers:{

            'Content-Type':'application/json',

            'Accept':'application/json',

        }

    })

}

Luego el post

import api from "./api";

export default{

    login(){

        const credencials={

            username:'admin',

            password:'password'

        }

        return api().post('/login',credencials);

    }

}

## Autenticación

Se puede modificar en el créate de axio o ponerlo como un valor por defecto en al gun momento, para que las siguientes peticiones tengan la autorización por defecto sin tener que estar agregándosela uno por uno

import axios from "axios";

const setAuthHeader=(token:string)=>{

    if (token){

        axios.defaults.headers.common['Authorization']='Bearer '+token;

    }else{

        delete axios.defaults.headers.common['Authorization'];

    }

}

export default setAuthHeader;