Momento 1:
crear e inicializar un nuevo proyecto de vue js , utilizando vue cli.

Plantillas con vuetify

¿Qué es Vuetify?

Vuetify es un marco de interfaz de usuario completo construido sobre Vue.js. El objetivo del proyecto es proporcionar a los desarrolladores las herramientas que necesitan para crear experiencias de usuario enriquecedoras y atractivas. A diferencia de otros marcos, Vuetify está diseñado desde cero para ser fácil de aprender y gratificante de dominar con cientos de componentes cuidadosamente elaborados a partir de la especificación de Material Design.

Vuetify adopta un enfoque de diseño que prioriza los dispositivos móviles, lo que significa que su aplicación simplemente funciona desde el primer momento, ya sea en un teléfono, tableta o computadora de escritorio.

Instalación vuetify:

vue add vuetify

```
C:\Users\andre\Desktop\prueba vuetify\vuetify-proyect>vue add vuetify
```

ahora nos aparece las características con las que vamos agregar vuetify a nuestro proyecto

```
Microsoft Windows [Versión 10.0.18363.1198]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\andre\Desktop\prueba vuetify\vuetify-proyect>vue add vuetify
 WARN  There are uncommitted changes in the current repository, it's recommended to commit or stash them first.
? Still proceed? (y/N) y
```

1. **elegir un preset**

```
  run `npm audit fix` to fix them, or `npm audit` for details
✔  Successfully installed plugin: vue-cli-plugin-vuetify

? Choose a preset:
  Default (recommended)
  Prototype (rapid development)
> Configure (advanced)
```

escogemos configuración avanzada damos tecla enter

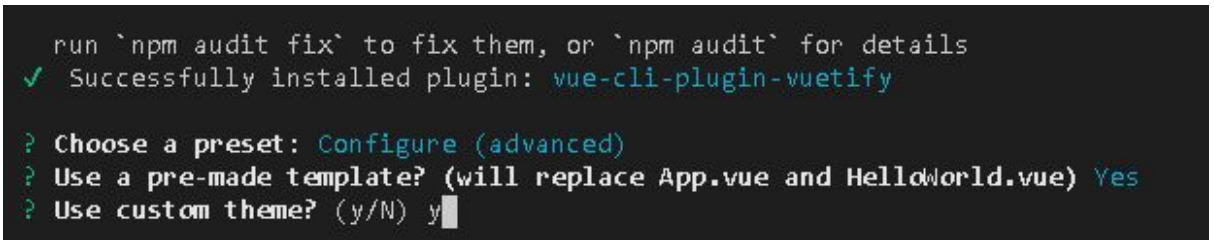
2. **utilizar una plantilla**

```
  run `npm audit fix` to fix them, or `npm audit` for details
✔  Successfully installed plugin: vue-cli-plugin-vuetify

? Choose a preset: Configure (advanced)
? Use a pre-made template? (will replace App.vue and HelloWorld.vue) (Y/n) y
```

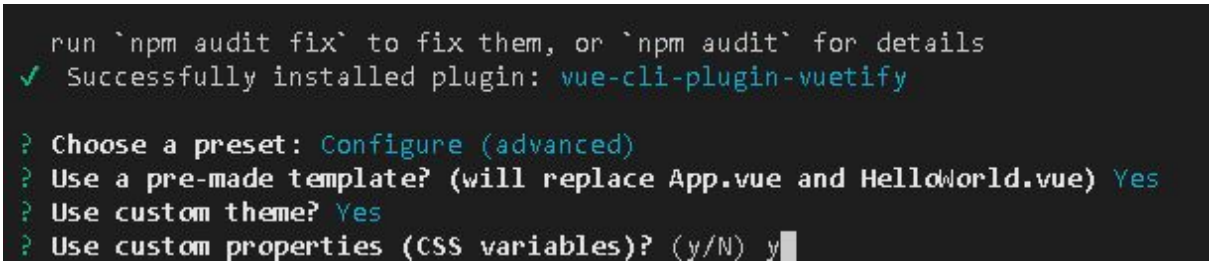
le damos y(yes) luego tecla enter

3. utilizar tema personalizado



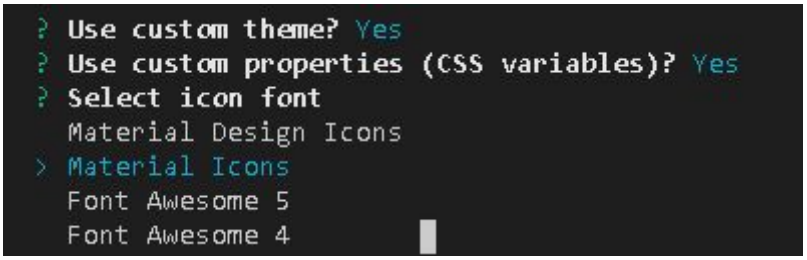
le damos y(yes) luego tecla enter

4. propiedades css



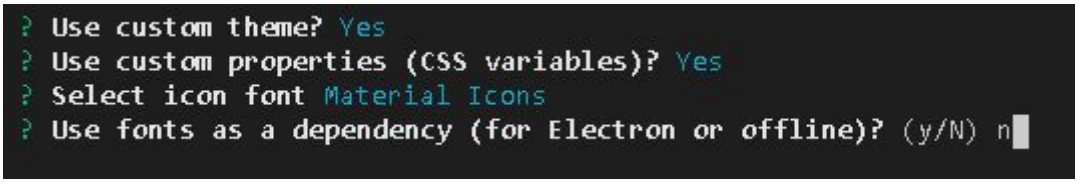
le damos y(yes) luego tecla enter

5. seleccionar fuente de icons



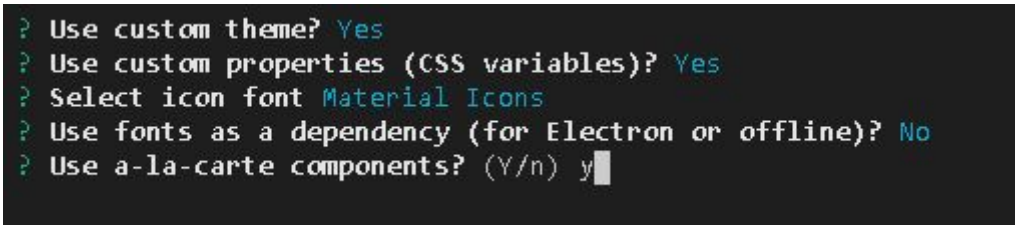
escogemos material icons y luego tecla enter

6. trabajar con electro



le damos n(no) y luego tecla enter

7. uso de componentes a la carta

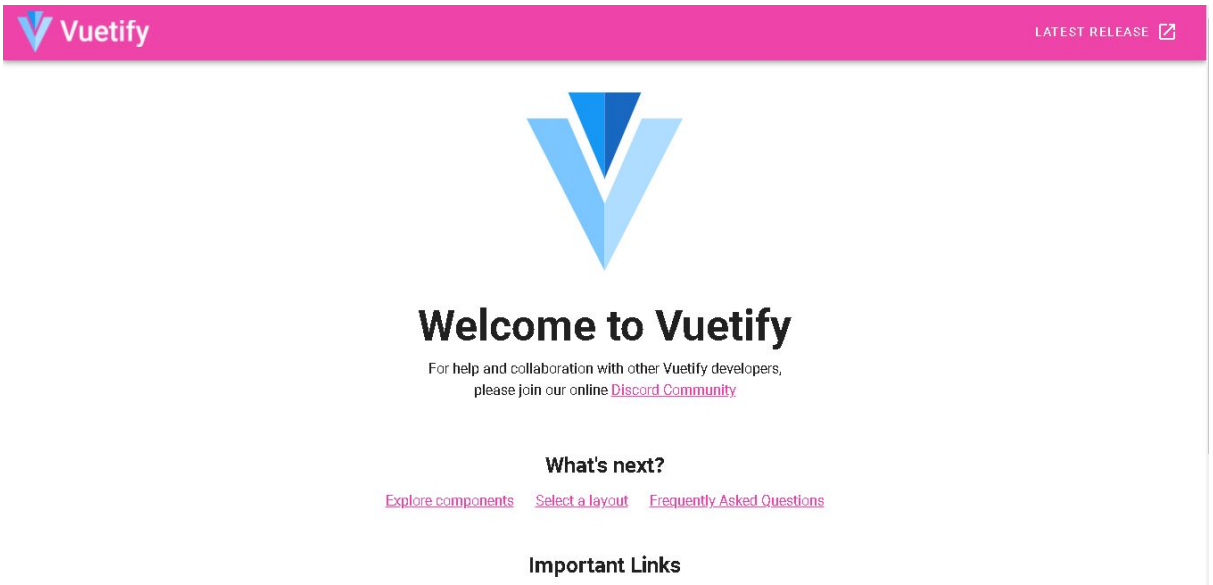


le damos y(yes) luego tecla enter

8. seleccionar localidad

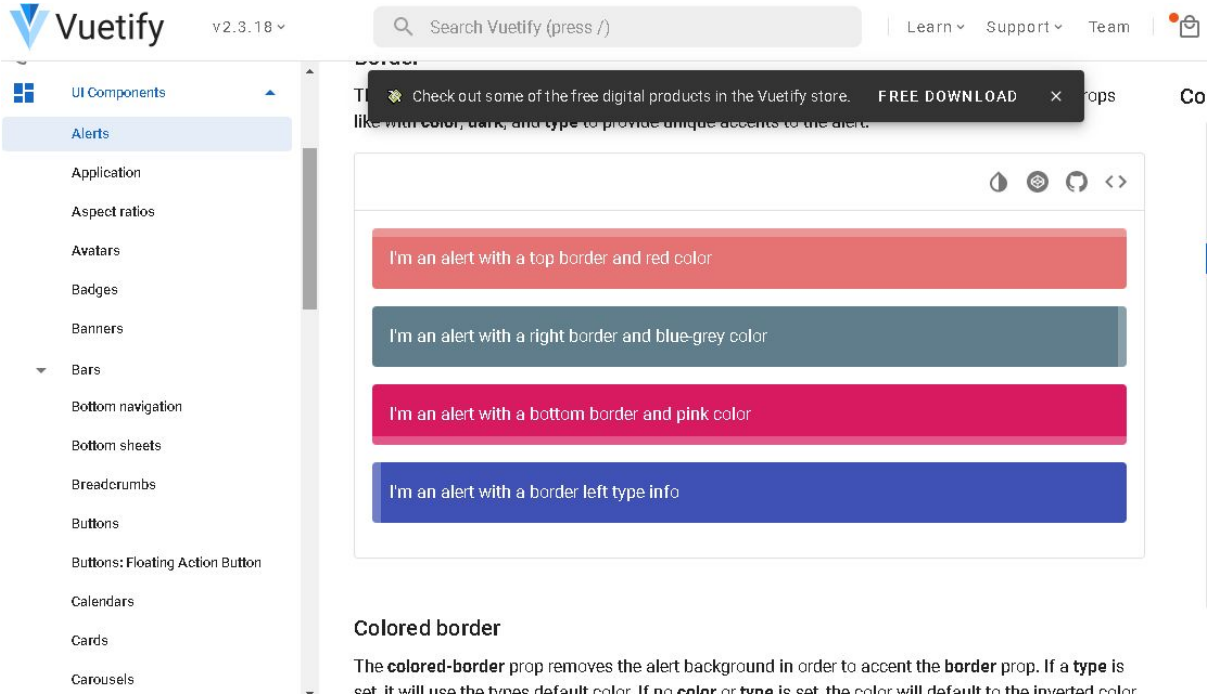
seleccionamos opción español

inicializamos nuestro servidor para comprobar que vuetify quedó instalado correctamente(npm run serve).



Momento 2:

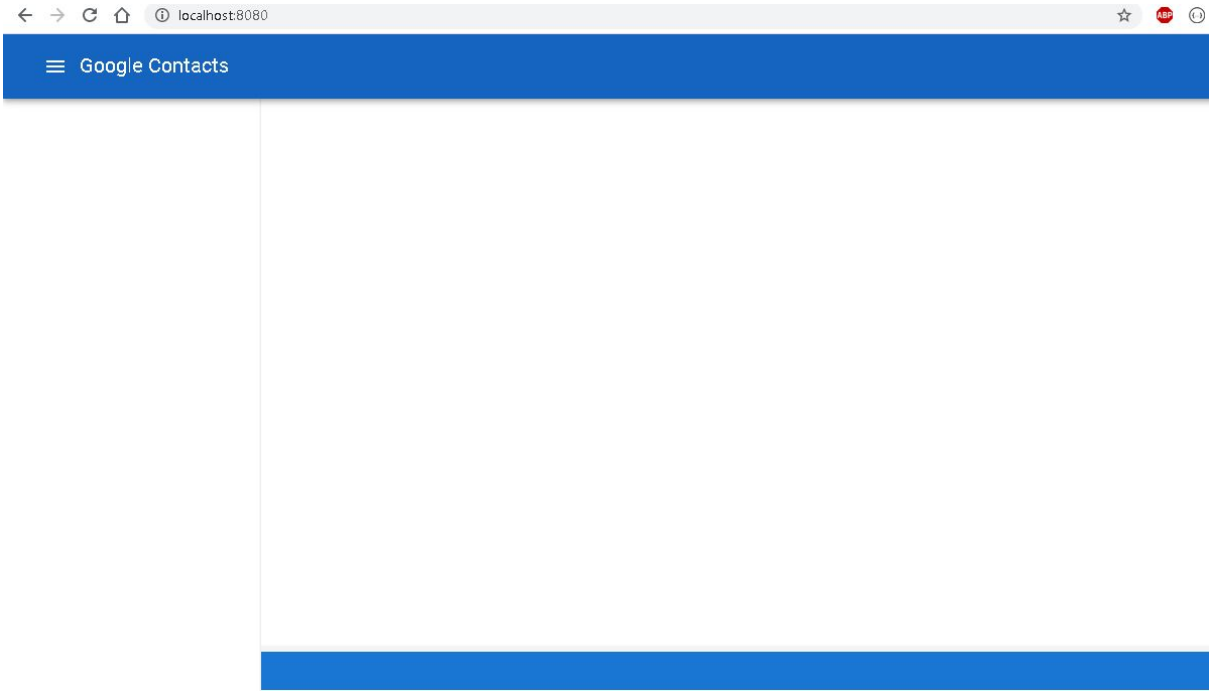
ingresamos a la página oficial de vuetify y podemos observar que encontramos diferentes componentes que podemos adaptar a nuestro proyecto según nuestras necesidades



que internamente ya traen unos estilos y animaciones que nos ahorran mucho tiempo de desarrollo .

Momento 3:
reutilización de componentes ya construidos, adaptándolo a nuestro proyecto:
en esta caso utilizaremos la una plantilla layouts de Google contacts, que se encuentra en la siguiente url:
https://github.com/Tecnalia-Cilco-3/Planillasvuetify-S4/blob/main/VuetifylayoutGoogleContacts.txt

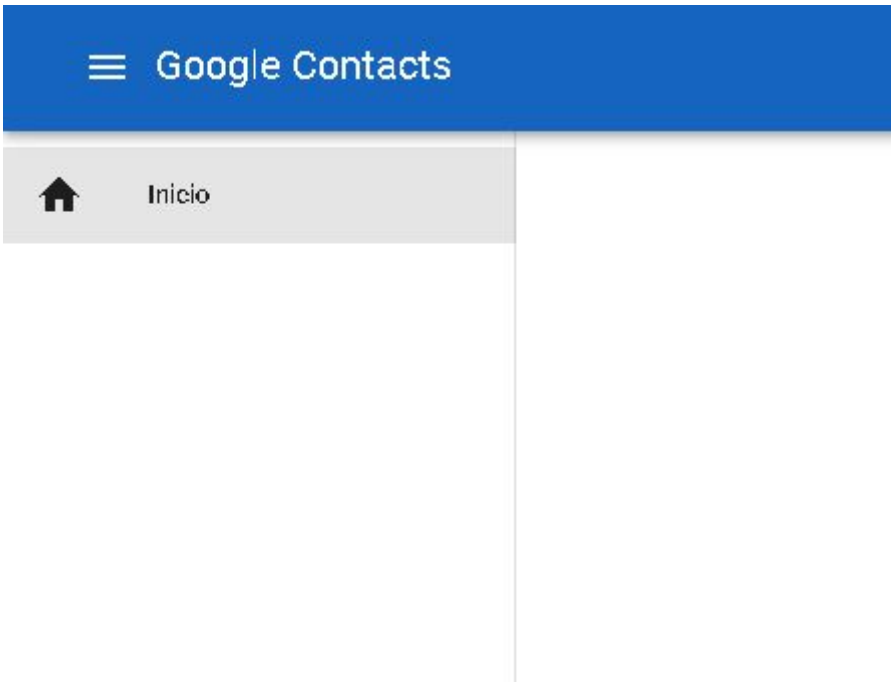
en nuestro proyecto nos ubicamos en el archivo App.vue eliminamos todos el contenido, pegamos el contenido que se encuentra en el la url de github y iniciamos nuestro servidor , si toda sale bien tenemos como resultado lo de la siguiente imagen:



modificando y agregando ítems del archivo App.js:

1. lo primero es cambiar el nombre de google contact , en este caso le pondremos sistema,nos ubicamos en el componente **v-toolbar-title>** buscamos donde dice *google contacts* lo modificamos por **sistema** y guardamos.
2. agregar opciones de menú de acceso
   lista items del menú:
   nos ubicamos dentro de la etiqueta **<v-list dense>**
   el primer menú que crearemos es el de inicio :
   abrimos una etiqueta <template/> dentro de ella crearemos el ítems inicio con su respectivo icono y título:

```
<template>
          <v-list-item :to="{name: 'home'}">
            <v-list-item-action>
              <v-icon>home</v-icon>
            </v-list-item-action>
            <v-list-item-title>
              Inicio
            </v-list-item-title>
          </v-list-item>
        </template>
```



agregamos otro **<template>** y dentro de él creamos un **<v-list-group> para poder visualizar opciones de menú con sub opciones**

```
<template>
          <v-list-group>
          </v-list-group>
</template>
```

y dentro de v-list-group agregamos :

```
<v-list-item slot="activator">
              <v-list-item-content>
                <v-list-item-title>
                  Almacén
                </v-list-item-title>
              </v-list-item-content>
            </v-list-item>
```

como lo hicimos item inicio pero al **v-list-item** le agregamos un **slot** activador esto para que nos despliegue el submenú y debajo del **</v-list-item> agregamos otros 2 nuevo <v-list-item> uno para categorías y otro para artículos con su respectivo título y icono**

```
 <v-list-item :to="{name: 'categoria'}">
              <v-list-item-action>
```

```
            <v-icon>table_chart</v-icon>
          </v-list-item-action>
          <v-list-item-content>
            <v-list-item-title>
              Categorías
            </v-list-item-title>
          </v-list-item-content>
        </v-list-item>
        <v-list-item :to="{name: ''}">
          <v-list-item-action>
            <v-icon>table_chart</v-icon>
          </v-list-item-action>
          <v-list-item-content>
            <v-list-item-title>
              Artículos
            </v-list-item-title>
          </v-list-item-content>
        </v-list-item>
```

el :to en el v-list-ome se refiere al nombre de la ruta que llamaremos el hacer click en cada submenú.

este mismo proceso lo realizamos para los otros submenús:

ventas:

- ventas
- clientes

accesos:

- usuario

funcionalidades crud en cada uno de nuestros componentes:
en este caso trabajaremos en **CRUD** categoría :

nos ubicamos en src/components y agregamos un nuevo componente llamado Categoria.vue agregamos la etiqueta <template>, este componente lo añadimos a el index.js que está ubicado en src/router/index con el **path**:/categorías,**name**:categorías. como hicimos con el módulo de layout buscaremos en la página oficial de vue un componente que adapte el manejo de crud en este caso: https://vuetifyjs.com/en/components/data-tables/#crud-actions.,dentro de la etiqueta template añadimos un <v-flex> y dentro de este empezaremos añadir el código de data tables en este caso de la documentación de vuetify. copiamos desde el <v-data-table>:

```
<template>
 <v-layout align-start>
    <v-flex>
        <v-data-table
      :headers="headers"
      :items="desserts"
      sort-by="calories"
      class="elevation-1"
    >
      <template v-slot:top>
        <v-toolbar
          flat
        >
          <v-toolbar-title>Categoria</v-toolbar-title>
          <v-divider
            class="mx-4"
            inset
            vertical
          ></v-divider>
          <v-spacer></v-spacer>
          <v-dialog
            v-model="dialog"
            max-width="500px"
          >
            <template v-slot:activator="{ on, attrs }">
              <v-btn
                color="primary"
                dark
                class="mb-2"
                v-bind="attrs"
                v-on="on"
              >
                New Item
              </v-btn>
            </template>
            <v-card>
```

```
            <v-card-title>
              <span class="headline">{{ formTitle }}</span>
            </v-card-title>

            <v-card-text>
              <v-container>
                <v-row>
                  <v-col
                    cols="12"
                    sm="6"
                    md="4"
                  >
                    <v-text-field
                      v-model="editedItem.name"
                      label="Dessert name"
                    ></v-text-field>
                  </v-col>
                  <v-col
                    cols="12"
                    sm="6"
                    md="4"
                  >
                    <v-text-field
                      v-model="editedItem.calories"
                      label="Calories"
                    ></v-text-field>
                  </v-col>
                  <v-col
                    cols="12"
                    sm="6"
                    md="4"
                  >
                    <v-text-field
                      v-model="editedItem.fat"
                      label="Fat (g)"
                    ></v-text-field>
                  </v-col>
                  <v-col
                    cols="12"
                    sm="6"
                    md="4"
                  >
                    <v-text-field
```

```
                    v-model="editedItem.carbs"
                    label="Carbs (g)"
                  ></v-text-field>
                </v-col>
                <v-col
                  cols="12"
                  sm="6"
                  md="4"
                >
                  <v-text-field
                    v-model="editedItem.protein"
                    label="Protein (g)"
                  ></v-text-field>
                </v-col>
              </v-row>
            </v-container>
          </v-card-text>

          <v-card-actions>
            <v-spacer></v-spacer>
            <v-btn
              color="blue darken-1"
              text
              @click="close"
            >
              Cancel
            </v-btn>
            <v-btn
              color="blue darken-1"
              text
              @click="save"
            >
              Save
            </v-btn>
          </v-card-actions>
        </v-card>
      </v-dialog>
      <v-dialog v-model="dialogDelete" max-width="500px">
        <v-card>
          <v-card-title class="headline">Are you sure you want to
delete this item?</v-card-title>
          <v-card-actions>
            <v-spacer></v-spacer>
```

```
                <v-btn color="blue darken-1" text
@click="closeDelete">Cancel</v-btn>
                <v-btn color="blue darken-1" text
@click="deleteItemConfirm">OK</v-btn>
                <v-spacer></v-spacer>
              </v-card-actions>
            </v-card>
          </v-dialog>
        </v-toolbar>
      </template>
      <template v-slot:[`item.actions`]="{ item }">
        <v-icon
          small
          class="mr-2"
          @click="editItem(item)"
        >
          mdi-pencil
        </v-icon>
        <v-icon
          small
          @click="deleteItem(item)"
        >
          mdi-delete
        </v-icon>
      </template>
      <template v-slot:no-data>
        <v-btn
          color="primary"
          @click="initialize"
        >
          Reset
        </v-btn>
      </template>
    </v-data-table>

    </v-flex>
 </v-layout>
</template>
```

como observamos lo anterior lo metimos desde un v-layout para ubicarlo en la parte superior y cambiamos el nombre de CRUD por categorías, New Item por

nuevo,básicamente es realizar modificaciones de textos de inglés a español y adaptarlos a nuestro proyecto.

agregamos un script en la parte inferior creamos nuestro objeto data , dentro ese data copiamos el objeto data del código fuente del datatable de la url anterior, también copiamos todos los otros métodos que se encuentran en script

```
<script>
export default {
    data(){
        return{
            dialog: false,
            dialogDelete: false,
            headers: [
            {
                text: 'Dessert (100g serving)',
                align: 'start',
                sortable: false,
                value: 'name',
            },
            { text: 'Calories', value: 'calories' },
            { text: 'Fat (g)', value: 'fat' },
            { text: 'Carbs (g)', value: 'carbs' },
            { text: 'Protein (g)', value: 'protein' },
            { text: 'Actions', value: 'actions', sortable: false },
            ],
            desserts: [],
            editedIndex: -1,
            editedItem: {
            name: '',
            calories: 0,
            fat: 0,
            carbs: 0,
            protein: 0,
            },
            defaultItem: {
            name: '',
            calories: 0,
            fat: 0,
            carbs: 0,
            protein: 0,
            }
        }
```

```
    },

     computed: {
        formTitle () {
        return this.editedIndex === -1 ? 'New Item' : 'Edit Item'
        },
    },

    watch: {
        dialog (val) {
        val || this.close()
        },
        dialogDelete (val) {
        val || this.closeDelete()
        },
    },

    created () {
        this.initialize()
    },

    methods: {
        initialize () {
        this.desserts = [
            {
            name: 'Frozen Yogurt',
            calories: 159,
            fat: 6.0,
            carbs: 24,
            protein: 4.0,
            },
        ]
        },

        editItem (item) {
        this.editedIndex = this.desserts.indexOf(item)
        this.editedItem = Object.assign({}, item)
        this.dialog = true
        },

        deleteItem (item) {
        this.editedIndex = this.desserts.indexOf(item)
        this.editedItem = Object.assign({}, item)
```

```
        this.dialogDelete = true
        },

        deleteItemConfirm () {
        this.desserts.splice(this.editedIndex, 1)
        this.closeDelete()
        },

        close () {
        this.dialog = false
        this.$nextTick(() => {
            this.editedItem = Object.assign({}, this.defaultItem)
            this.editedIndex = -1
        })
        },

        closeDelete () {
        this.dialogDelete = false
        this.$nextTick(() => {
            this.editedItem = Object.assign({}, this.defaultItem)
            this.editedIndex = -1
        })
        },

        save () {
        if (this.editedIndex > -1) {
            Object.assign(this.desserts[this.editedIndex],
this.editedItem)
        } else {
            this.desserts.push(this.editedItem)
        }
        this.close()
        },
    },
}
</script>
```

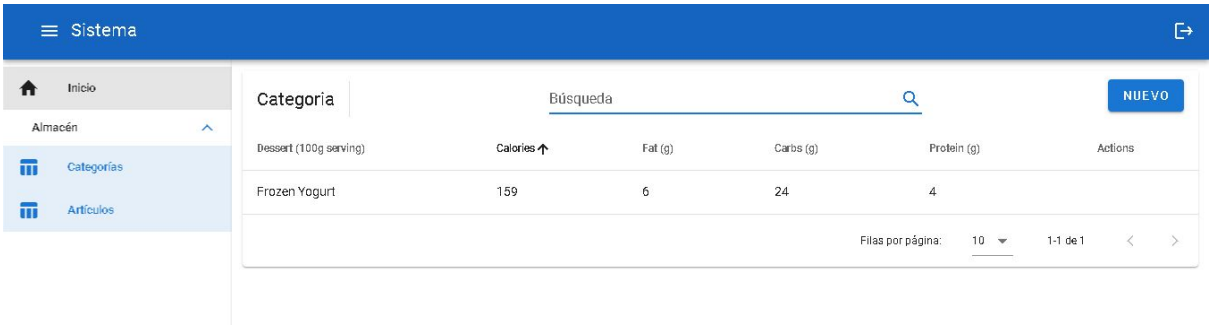
ahora lo que haremos es añadir un buscador,añadimos la siguientes lineas de código

```
<v-text-field class="text-xs-center" v-model="search"
append-icon="search"
                label="Búsqueda" single-line
hide-details></v-text-field>
                <v-spacer></v-spacer>
```

lo agregamos en el v-toolbar debajo de las líneas <v-spacer></v-spacer> creamos la variable search(la inicializamos en vacío ‘’) en nuestro objeto data y la relacionamos por medio del v-model, luego vamos al datatable y agregamos :search=”search” que esta relacionada con el v-text-field

```
<v-data-table
            :headers="headers"
            :items="desserts"
            :search="search"
            sort-by="calories"
            class="elevation-1"
            >
```

guardamos , vamos a nuestro navegar y si todo sale bien obtenemos el siguiente resultado:



realizamos búsquedas en input para probar que esté funcionando correctamente.

1. **Lista, búsqueda y paginación con los datos de nuestra bases de datos usando axios (npm install axios):**

listado:

declaramos una array vacío en objeto data llamado categorías donde almacenaremos todos los registros que nos devuelva el backend por medio de la petición mediante axios:

crearemos un método llamado listar que lo único que vas hacer que me permite obtener el listado de todas las categorías:

```
listar(){
            let me=this;
            axios.get('categoria/list',configuracion).then(function (response){
               me.categorias=response.data;
            }).catch(function(error){
               console.log(error);
            });
                  },
```

en el hook created llamaremos a este método listar para que sea el primero en ejecutarse e inicializar la array categorías , eliminaremos el que venía en la planilla con su objeto en el data.

Nota : para añadir un url base nos ubicamos en el archivo main.js importamos axios y añadimos la siguiente línea de código:

```
import axios from 'axios';

Vue.config.productionTip = false;
axios.defaults.baseURL = 'http://localhost:3000/api/';
```

esto con el fin de que al momento de un cambio en la url para la petición de datos , no tengamos que modificar archivo por archivo.

el siguiente paso es modificar la cabecera del datatable vamos al objeto data y cambiar el contenido del array head por el siguiente :

```
headers: [
            { text: 'Opciones', value: 'opciones', sortable: false },
            { text: 'Nombre', value: 'nombre', sortable: true },
            { text: 'Descripción', value: 'descripcion', sortable: false },
            { text: 'Estado', value: 'estado', sortable: false },
        ],
```

donde el componente datatable renderiza ese objeto, donde text es el nombre de la columna,value es el valor del atributo del objeto categoría , sortable:false que no esté ordenado.

2. **crear una nueva categoría**
   crearemos nuevas variables en el objeto data,_id,nombre,descripcion, para relacionarla con el v-model, que se encuentran dentro de él v-dialog,tenemos otra variable dialog que venía en la planilla que según su valor,true o false se abrirá el modal:

   ```
   <v-dialog v-model="dialog" max-width="500px">
   ```

   el objeto data debe estar actualizado como el siguiente fragmento del código:

```
    data(){
    return{
      dialog: false,
      search:'',
      categorias:[],
      headers: [
        { text: 'Opciones', value: 'opciones', sortable: false },
        { text: 'Nombre', value: 'nombre', sortable: true },
        { text: 'Descripción', value: 'descripcion', sortable: false },
        { text: 'Estado', value: 'estado', sortable: false },
      ],
```

```
            editedIndex: -1,
            _id:'',
            nombre:'',
            descripcion:'',
}
```

al relacionar con los v-model nos quedaria asi :

```
<v-dialog v-model="dialog" max-width="500px">
              <template v-slot:activator="{ on }">
                <v-btn color="primary" dark class="mb-2" v-on="on">Nuevo</v-btn>
              </template>
              <v-card>
                <v-card-title>
                <span class="headline">{{ formTitle }}</span>
                </v-card-title>
                <v-card-text>
                <v-container grid-list-md>
                  <v-layout wrap>
                    <v-flex xs12 sm12 md12>
                      <v-text-field v-model="nombre" label="Nombre"></v-text-field>
                    </v-flex>
                    <v-flex xs12 sm12 md12>
                      <v-text-field v-model="descripcion" label="Descripción"></v-text-field>
                    </v-flex>
                    <v-flex xs12 sm12 md12 v-show="valida">
                      <div class="red--text" v-for="v in validaMensaje" :key="v" v-text="v">

                      </div>
                    </v-flex>
                  </v-layout>
                </v-container>
                </v-card-text>
                <v-card-actions>
                <v-spacer></v-spacer>
                <v-btn color="blue darken-1" flat @click="close">Cancelar</v-btn>
                <v-btn color="blue darken-1" flat @click="guardar">Guardar</v-btn>
                </v-card-actions>
              </v-card>
            </v-dialog>
```

como vemos tenemos dos eventos que son close para cancelar y guardar
método guardar:

```
guardar(){
            let me=this;
            let header={"Token" : this.$store.state.token};
            let configuracion= {headers : header};
```

```
            if (this.validar()){
                return;
            }
            if (this.editedIndex >-1){
                //Código para editar

axios.put('categoria/update',{'_id':this._id,'nombre':this.nombre,'descripcion':this.descripcion},confi
guracion)
                .then(function(response){
                    me.limpiar();
                    me.close();
                    me.listar();
                })
                .catch(function(error){
                    console.log(error);
                });
            }else{
                //Código para guardar

axios.post('categoria/add',{'nombre':this.nombre,'descripcion':this.descripcion},configuracion)
                .then(function(response){
                    me.limpiar();
                    me.close();
                    me.listar();
                })
                .catch(function(error){
                    console.log(error);
                });
            }
        },
```

explicación método guardar:
lo primero que hacemos es capturar el token que lo explicaremos más adelante cómo almacenarlo en el storage del navegador y darle manejo,
luego llamaremos a otro método validar:
donde determinaremos si en los text-field si el usuario a ingresado un texto o no , si no a ingresado nada le mostraremos un mensaje , y para esto declaramos dos variables en nuestro objeto data el primero es valida:0 si el valor de esta variable cambia a uno mostramos el mensaje de lo contrario el mensaje no se mostrará , la otra variable es de tipo array
validaMensaje=[];, le insertamos los mensajes de validación si el usuario no llena completamente el formulario.

```
validar(){
            this.valida=0;
            this.validaMensaje=[];
            if(this.nombre.length<1 || this.nombre.length>50){
```

```
            this.validaMensaje.push('El nombre de la categoría debe tener entre 1-50 caracteres.');
          }
          if(this.descripcion.length>255){
            this.validaMensaje.push('La descripción de la categoría no debe tener más de 255 caracteres.');
          }
          if (this.validaMensaje.length){
            this.valida=1;
          }
          return this.valida;
        },
```

para mostrar este mensaje agregamos otro textfield con la etiqueta v-show que renderiza la variable valida segun su valor:

```
<v-flex xs12 sm12 md12 v-show="valida">
                    <div class="red--text" v-for="v in validaMensaje" :key="v" v-text="v">

                    </div>
                  </v-flex>
```

luego en nuestro método guardar validamos si el evento es de guardar o de actualizar y esto lo hacemos por medio de la variable editedIndex que viene en la planilla utilizada, la cual nos dice que si esta variable es igual -1 es por se ha presionado la opción de crear nueva categoría pero sí de lo contrario es 1 la que se quiere es editar , dependiendo de la opción ,ya sea update o create ,realizaremos una petición por axios a la ruta correspondiente para dicha acción ,la acción create enviamos nombre y descripción ya que son campos obligatorios en nuestra bd y el objeto configuración con el token ,para que pase el middleware de autenticación del backend ,si la petición devuelve un estatus de aceptación,limpiaremos los text fiel ,los mensajes de validación los ocultaremos y los borramos ,reiniciamos la variable editedindex. para eso creamos un nuevo metodo llamado limpiar:

```
 limpiar(){
                this._id='';
                this.nombre='';
                this.descripcion='';
                this.valida=0;
                this.validaMensaje=[];
                this.editedIndex=-1;
            },
```

cerramos el modal con el método close :

```
close () {
                this.dialog = false;
```

```
            },
```

llamaremos el método list de nuevo para actualizar nuestra tabla de categorías.

si la acción es de actualizar(update) el valor de la variable iditedindex tendrá el valor de 1,pero primero tendremos que añadir un evento que el usuario pueda accionar para realizar la modificación, para esto instalaremos un icono en la columna acciones que cuando el usuario le de click se abre el modal , insertamos el siguiente fragmento de código abajo de nuestor data table

```
<v-data-table
                :headers="headers"
                :items="categorias"
                :search="search"
                class="elevation-1"
            >
            <template v-slot:[`item.opciones`]="{ item }">
                <v-icon
                    small
                    class="mr-2"
                    @click="editItem(item)"
                    >
                    edit
                    </v-icon>
                        <v-icon v-if="item.estado"
                        small
                        @click="activarDesactivarMostrar(2,item)"
                        >
                        block
                        </v-icon>

                        <v-icon v-else
                        small
                        @click="activarDesactivarMostrar(1,item)"
                        >
                        check
                        </v-icon>
            </template>
```

:

```
<template v-slot:[`item.opciones`]="{ item }">
```

v-slot:[`item.opciones`]="{ item }" nos ubica en la columna opciones y le pasamos el item seleccionado que es alguna de las categorías,con el componente v-icon creamos un icono

de editar y le pasamos el evento @click que invoca la función "editItem(item)" con el item seleccionado, el método edititem los modificaremos, lo que haremos es enviar los valores que tenemos en el objeto items a las variables de objeto data que son _id,nombre,descripcion , y le tenemos que decir que habrá el modal con dialog=true , editedindex=1 para así entrar en el método guardar en el if que realiza el update como vimos anteriormente:

```
editItem (item) {
                this._id=item.id;
                this.nombre=item.nombre;
                this.descripcion=item.descripcion;
                this.dialog = true;
                this.editedIndex=1;
            },
```

por último queremos que el usuario pueda cambiar el estado de una categoría para cuando esta no se encuentre disponible,como podemos observar en el fragmento de código donde creamos el icon de editar, debajo de este creamos otros dos uno de check y otro de block , y ponemos la etiqueta v-if,v-else, para verificar el estado de la categoría,y así determinar si mostraremos el icono de check o block.

para poder actualizar el estado de la categoría debemos declarar un evento en cada icono,@click="activarDesactivarMostrar(1,item)",
lo que primero hacemos es declarar nuevas variables en objeto data , ya que también **queremos un modal de confirmación del cambio de estado:**

adModal:0, si el se pone en 1 el modal se mostraria.
adAccion:0, para determinar si la acción es de activar o desactivar.
adNombre:", el nombre de la categoría para la confirmación del usuario.
adId:"" , el id del item para cuando realicemos el update a la base de datos.

lo primero es crear un modal que ya sabemos que es utilizando el componente dialog

```
<v-dialog v-model="adModal" max-width="290">
            <v-card>
              <v-card-title class="headline" v-if="adAccion==1">
                Activar Item
              </v-card-title>
              <v-card-title class="headline" v-if="adAccion==2">
                Desactivar Item
              </v-card-title>
              <v-card-text>
                Estás a punto de <span v-if="adAccion==1">activar </span>
```

```
                    <span v-if="adAccion==2">desactivar </span> el item {{adNombre}}
                </v-card-text>
                <v-card-actions>
                    <v-spacer></v-spacer>
                    <v-btn @click="activarDesactivarCerrar()" color="green darken-1" flat="flat">
                        Cancelar
                    </v-btn>
                    <v-btn v-if="adAccion==1" @click="activar()" color="orange darken-4"
flat="flat">
                        Activar
                    </v-btn>
                    <v-btn v-if="adAccion==2" @click="desactivar()" color="orange darken-4"
flat="flat">
                        Desactivar
                    </v-btn>
                </v-card-actions>
            </v-card>
        </v-dialog>
```

es básicamente realizar validación del estado de las variables que se crearon anteriormente y según ese estado realizamos otros eventos ya sea ,activar o desactivar , estos dos métodos lo que harán es realizar un update (que ya lo hicimos anteriormente)pero solo modificará el estado de dicha categoría según el id.

```
activarDesactivarCerrar(){
            this.adModal=0;
        },
        activar(){
            let me=this;
            let header={"Token" : this.$store.state.token};
            let configuracion= {headers : header};

            axios.put('categoria/activate',{'_id':this.adId},configuracion)
                .then(function(response){
                    me.adModal=0;
                    me.adAccion=0;
                    me.adNombre='';
                    me.adId='';
                    me.listar();
                })
                .catch(function(error){
                    console.log(error);
```

```
                });
        },
        desactivar(){
            let me=this;
            let header={"Token" : this.$store.state.token};
            let configuracion= {headers : header};
            axios.put('categoria/deactivate',{'_id':this.adId},configuracion)
                .then(function(response){
                    me.adModal=0;
                    me.adAccion=0;
                    me.adNombre='';
                    me.adId='';
                    me.listar();
                })
                .catch(function(error){
                    console.log(error);
                });
        },
        close () {
            this.dialog = false;
        },


    }
```

y cada que hagamos estas peticiones y su respuesta sea satisfactoria tenemos que limpiar las variables.

el método activarDesactivarCerrar, lo ponemos como evento en el botón cancelar del modal ,con la función de cerrar ese modal poniendo la variable admodal en 0

```
activarDesactivarCerrar(){
            this.adModal=0;
        },
```

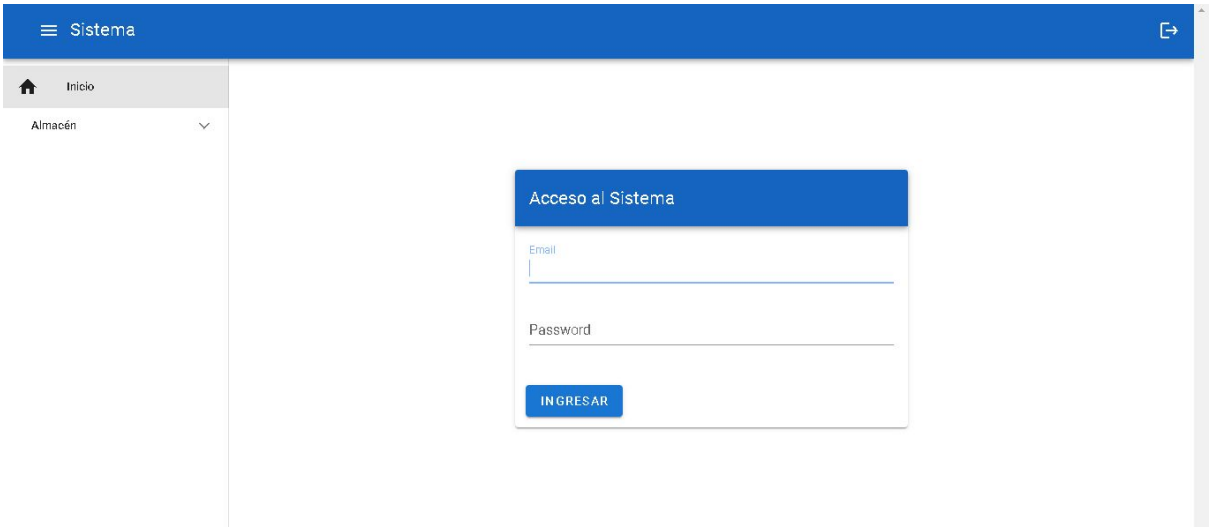
Para realizar los otro componentes faltantes podemos reutilizar el mismo código utilizado para categorías pero teniendo en cuenta sus atributos y la ruta definida para ese modelo en el backend.

migración de login de la sesion 3 a vuetify

añadiremos la etiqueta template recordemos que aquí va el código html,vamos a trabajar en el código html para diseñar el formulario para el ingreso al sistema.

primero es agregar una etiqueta v-layout para alienar formulario en el centro y justificar el contenido, y dentro de este agregamos una etiqueta v-flex para indicarle que medida va tener según los dispositivos y dentro de este ingresamos el contenido que tendrá nuestro formulario

```
<template>
  <v-layout align-center justify-center>
    <v-flex xs12 sm8 md6 lg5 x14>
      <v-card>
        <v-toolbar dark color="blue darken-3">
          <v-toolbar-title>
            Acceso al Sistema
          </v-toolbar-title>
        </v-toolbar>
        <v-card-text>
          <v-text-field  autofocus color="accent" label="Email" required>
          </v-text-field>
          <v-text-field  type="password" color="accent" label="Password" required>
          </v-text-field>
          <v-flex class="red--text" v-if="error">
            errores
          </v-flex>
        </v-card-text>
        <v-card-actions class="px-3 pb-3">
          <v-flex text-xs-right>
            <v-btn color="primary">Ingresar</v-btn>
          </v-flex>
        </v-card-actions>
      </v-card>
    </v-flex>
  </v-layout>
</template>
```



creamos la ruta login:

```
{
        path: "/login",
        name: "login",
        component: Login
    },
```

volvemos a nuestro componente Login.vue y editaremos el script , agregamos 3 variables en el objeto data que son email,password que las inicializamos en vacío y ErrorM que la inicializamos en null

```
export default {
    data(){
        return{
            email:'',
            password:'',
            errorM:null
        }
    }

}
</script>
```

ahora debemos relacionar estas variables con una etiqueta v-model en el formulario:

```
<template>
  <v-layout align-center justify-center>
    <v-flex xs12 sm8 md6 lg5 x14>
      <v-card>
        <v-toolbar dark color="blue darken-3">
          <v-toolbar-title>
```

```
                Acceso al Sistema
              </v-toolbar-title>
            </v-toolbar>
            <v-card-text>
              <v-text-field v-model="email" autofocus color="accent" label="Email" required>
              </v-text-field>
              <v-text-field v-model="password" type="password" color="accent" label="Password" required>
              </v-text-field>
              <v-flex class="red--text" v-if="errorM">
                {{errorM}}
              </v-flex>
            </v-card-text>
            <v-card-actions class="px-3 pb-3">
              <v-flex text-xs-right>
                <v-btn  color="primary">Ingresar</v-btn>
              </v-flex>
            </v-card-actions>
          </v-card>
        </v-flex>
      </v-layout>

</template>
<script>
```

lo siguiente es enviar los datos al backend para validar y recibir el token de autenticación , que recordemos que el token lo encriptamos con el id, rol del usuario y texto clave,para esto definimos un evento en botón ingresar

```
<v-btn @click="ingresar()" color="primary">Ingresar</v-btn>
```

como ya sabemos debemos definir un método llamado ingresar

```
ingresar(){
        axios.post('usuario/login',{email: this.email, password: this.password})
        .then(respuesta =>{
            return respuesta.data;
        })
        .then(data =>{
        })
        .catch(error =>{
            //console.log(eror);
            this.errorM=null;
            if (error.response.status==404){
                this.errorM='No existe el usuario o las credenciales son incorrectas.';
            } else{
                this.errorM='Ocurrió un error con el servidor.';
```

```
        }
      });
    }
```

lo primero que hacemos es realizar un petición axios por medio del método post al back en a la ruta
usuario/login y le enviamos las variables email y password y esperamos una respuesta que como sabemos que es un token con información encriptada del usuario, si el servidor responde un error lo manejaremos con condicionales para detectar lo sucedido.

ahora nos ubicamos en el directorio store/index.js para poder gestionar el acceso y la autorización al sistema, pero para poder decodificar la firma generada en el backen (token) debemos utilizar json web token , si no lo tenemos descargado abrimos la terminal integrada si estamos utilizando vs code y ejecutamos el siguiente comando:
npm install jwt-decode
y lo importamos en el archivo index.js(directorio store).

```
import decode from "jwt-decode";
```

y también nuestras rutas:

```
import router from "../router/index";
```

el archivo tienes 3 objetos :

- state:no puede ser modificado directamente es de solo lectura
- mutations:mutar lógica para cambiar el estado
- action: se declaran acciones que son similares a las mutaciones pero con la diferencia que en lugar de mutar al state , las acciones realizan mutacion , es decir realizan funciones asíncronas arbitrarias

dentro del objeto state vamos a declarar dos variables que las inicializamos en null token y usuario
donde:

- token: almacenaremos el token
- usuario:almacenaremos el usuario

```
state: {
        token: null,
        usuario: null
    },
```

dentro del objeto mutations vamos definir los métodos que van a modificar el contenido de las variables definidas en objeto stata

- setToken:va esperar el state y el token , para igualar el token del state ,pro el token recibido
- getUsuario:va esperar el state y el usuario, para igualar el usuario del state con el usuario
  recibido
-

```
mutations: {
        setToken(state, token) {
```

```
            state.token = token;
        },
        setUsuario(state, usuario) {
            state.usuario = usuario;
        }
    },
```

dentro del objeto action vamos a definir 3 acciones :

- guardarToken:me va permitir que ese token que devuelto en el bakend almacenarlo en el localstorage del navegador
- autoLogin: me va verificar si el usuario ya tiene un token válido en el local storage ya no va pedir que nuevamente ingrese al sistema, por ejemplo al refrescar la página.
- salir: elimina el token del local storage y finaliza sesion el usuario

```
actions: {
      guardarToken({ commit }, token) {
        commit("setToken", token)
        commit("setUsuario", decode(token))
        localStorage.setItem("token", token)
      },
      autoLogin({ commit }) {
        let token = localStorage.getItem("token");
        if (token) {
          commit("setToken", token);
          commit("setUsuario", decode(token));
        }
            },
      salir({ commit }) {
        commit("setToken", null);
        commit("setUsuario", null);
        localStorage.removeItem("token");
        router.push({ name: 'login' });
      }
  },
```

el commit permite realizar una mutación al state.

ahora nos dirigimos al directorio router/index.js

a las rutas que solo podrán acceder con autenticación le agregaremos un nuevo parámetro llamado meta :

```
meta: {
        auth: true
      }
```

si no necesitan autenticación le agregamos el meta:

meta: {
libre: true
}

Cuando se llama a la ruta , llama a un método router.beforeEach que verifica en el store (en el archivo index.js) objeto state si existe un usuario que sea diferente de null , si existe un lo dejamos pasar , de resto lo redirigimos al login.

```
import Vue from "vue";
import VueRouter from "vue-router";
import Home from "../views/Home.vue";
import Categoria from '../components/Categoria.vue';
import Login from '../components/Login.vue';
import store from '../store/index';

Vue.use(VueRouter);

const routes = [{
        path: "/",
        name: "home",
        component: Home,
        meta: {
            public: true

        }

    },
    {
        path: "/login",
        name: "login",
        component: Login,
        meta: {
            public: true

        }
    },
    {
        path: "/categoria",
        name: "categoria",
        component: Categoria,
        meta: {
            auth: true
        }

    },
```

```
];

const router = new VueRouter({
    mode: "history",
    base: process.env.BASE_URL,
    routes
});

router.beforeEach((to, from, next) => {
    if (to.matched.some(record => record.meta.public)) {
        console.log("hola");
        next();
    } else if (store.state.usuario) {
        if (to.matched.some(record => record.meta.auth)) {
            console.log(store.state.usuario);
            next();
        }
    } else {
        next({ name: 'login' });
    }
})

export default router;
```

siguiente paso nos dirigimos al App.vue es crear dos métodos uno en el hook computed y el otro en el hook methods

en el hook computed creamos el metodo login : lo que hará es verificar si en nuestro store existe un usuario por lo que retorna falso o verdadero

```
logueado(){
      return this.$store.state.usuario;
    },
```

hook methods creamos el método salir que lo único que hace es que nuestro store ejecuta la acción salir que hacer referencia por medio dispatch, recordemos que que la acción salir lo que hace es remover el token y redirigir al usuario al login;

```
methods:{
    salir(){
      this.$store.dispatch("salir");
    }
  }
```

por último en el hook create de nuevo vamos nuestro store hacemos referencia a la acción autologin por si el usuario refresca haga mutación a las variables del state

```
created(){
    this.$store.dispatch("autoLogin");
  }
```

ahora en el templete de categorías realizamos una validación para saber si el usuario está autenticado , si el método logueado retorna true el usuario visualiza el menú categorías de lo contrario quedará oculto para este usuario.

```
<template v-if="logueado">
          <v-list-group>
            <v-list-item slot="activator">
              <v-list-item-content>
                <v-list-item-title>
                  Almacén
                </v-list-item-title>
              </v-list-item-content>
            </v-list-item>
            <v-list-item :to="{name: 'categoria'}">
              <v-list-item-action>
                <v-icon>table_chart</v-icon>
              </v-list-item-action>
              <v-list-item-content>
                <v-list-item-title>
                  Categorías
                </v-list-item-title>
              </v-list-item-content>
            </v-list-item>
            <v-list-item :to="{name: ''}">
              <v-list-item-action>
                <v-icon>table_chart</v-icon>
              </v-list-item-action>
              <v-list-item-content>
                <v-list-item-title>
                  Artículos
                </v-list-item-title>
              </v-list-item-content>
            </v-list-item>
          </v-list-group>
        </template>
```

ahora en el botón con el icono logout crearemos un evento para ejecutar el método salir pero eso solo se mostrará si el usuario se encuentra logueado

```
<v-btn @click="salir()" icon v-if="logueado">
        <v-icon>logout</v-icon> Salir
      </v-btn>
```

.

plantillas para componente home:
https://next.vuetifyjs.com/en/components/data-iterators/#filter