**1) Crear el componente con la estructura básica**

Estructura básica de un componente

```vue
<template>
  <div></div>
</template>

<script>
export default {
  name: "ComponentPage",
  created() {},
  data() {
    return {};
  },
  props: {},
  methods: {},
};
</script>
```

**2) Agregar la ruta correspondiente al componente**

***Las rutas padre ya existen

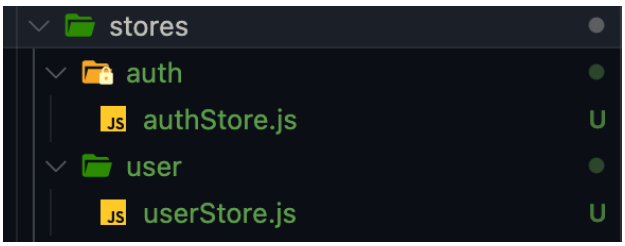
```
{
  path: '/contabilidad',
  component: () => import('layouts/MainLayout.vue'),
  children: [
    {
      name: "pagos", path: 'pagos', component: () => import('pages/contabilidad/pagos.vue')
    }
  ]
},
```

La nueva ruta de agrega de acuerdo a lo que contienen las llaves amarillas(dentro de la propiedad “children”)

**3) Crear el store**

**Existen 2 stores(auth: se usa para autenticación, user: se usa para la gestión de usuarios)

Cada store debe estar dentro de un directorio, ejemplo:



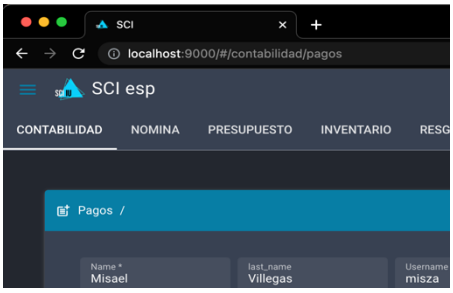
Estructura de un store


Prácticamente solo se debe cambiar la palabra “user“ de acuerdo al módulo en el que se esté trabajando(factura, pago, incidencia, etc.).

Estructura HTML invocando a GenericTable

```
<template>
  <q-page padding>
    <GenericTable
      v-if="getterData.length > 0"
      ref="child"
      :table-title="'Pagos'"
      :form-config="formConfig"
      :title-export="'pagos'"
      :getter-data="getterData"
      :api-route="'users/'"
      :front-route="'/contabilidad/pagos'"
      v-on:sync:data="syncData($event)"
      v-on:send:put="putRecord($event)"
      v-on:send:post="postRecord($event)"
      v-on:send:del="deleteRecord($event)"
    />
  </q-page>
</template>
```

Table-title: nombre de la tabla.
Title-export: nombre del archivo csv que se genera al exportar los datos.
Api-route: nombre de la ruta padre que gestionará el componente.
Front-route: nombre de la ruta que aparece en la barra de búsqueda



Estructura JavaScript

```vue
<script>
import { defineComponent } from "vue";  158.1k (gzipped: 58.1k)
import GenericTable from "src/components/custom/GenericTable.vue";

import { useUserStore } from "src/stores/user/userStore";
export default defineComponent({…
});
</script>
```

Configuración del componente

```vue
export default defineComponent({
  name: "PagosPage",
  components: {
    GenericTable,
  },
  setup() {
    const userStore = useUserStore();
    return {
      userStore,
    };
  },
  data: function () {
    return {
      data: [],
      getterData: [],
      formConfig: [
        {
          element: "name",
          type: "text",
          required: true,
          label: "Nombre",
        },
        {
          element: "last_name",
          type: "text",
          required: false,
        },
        {
          element: "username",
          type: "text",
          required: true,
        },
        {
          element: "email",
          type: "text",
          required: false,
        },
      ],
    };
  },
```

**Name**: nombre del componente, debe conformarse por 2 palabras.
**Setup**: es un método nativo de vue, se utiliza para configurar variables o constantes
**userStore**: invoca al manejador de estados que utilizará el componente, en este caso es el de usuarios.
**Data**: lugar en donde se definen valores por default que irán cambiando.
**getterData**, **formConfig**: obligatorios.
**formConfig**: lugar en donde se define la configuración del formulario.

- Element: nombre del campo en la BD
- Type: tipo de input(text, select, combo, number)
- Required: establecer en true o false dependiendo las restricciones de la BD
- Label: texto que se mostrará en el input

Los siguientes métodos gestionan el crud
**getData**:

- Utiliza el store importado para listar la información de la BD y guardarla en el propio store.
- La parte que de debe configurar se ubica dentro del “try”, la primera linea ejecuta la petición al servidor, la segunda asigna el resultado de lo anterior para ser enviado a la tabla.

putRecord:

- Se encarga de actualizar un registro.
- La parte que se debe configurar es la que inicia con “await”, se debe cambiar “this.userStore.putUser” por la función que existe en el store correspondiente, por ejemplo “this.inventarioStore.putElemento”.

postRecord:

- Se encarga de guardar un nuevo registro.
- Se debe configurar al igual que el anterior método, obviando que en vez de “putElemento” es “postElemento”

deleteRecord:

- Se encarga de eliminar el registro.
- Se debe configurar al igual que los anteriores metodos.

```
mounted() {
  this.getData();
},
methods: {
  async getData() {
    try {
      await this.userStore.fetchUsers();
      this.getterData = this.userStore.getUsers;
    } catch (err) {
      console.log(err);
      if (err.response.data.error) {
        $q.notify({
          type: "negative",
          message: err.response.data.error,
        });
      }
    }
  },
  async putRecord(ev) {
    console.log(ev);
    try {
      await this.userStore.putUser(ev.rows.id, ev.formData);
    } catch (error) {
      console.error(error);
    }
  },
  async postRecord(ev) {
    try {
      await this.userStore.postUser(ev);
      this.$refs.child.cancel();
    } catch (error) {
      if (error.response.data.errors) {
        let msg = error.response.data.errors;
        let keys = Object.keys(msg);
        console.log(keys);
        for (let index = 0; index < keys.length; index++) {
          this.$q.notify({
            type: "negative",
            position: "bottom",
            message: `${keys[index].toUpperCase()}: ${msg[keys[index]]}`,
          });
        }
      }
    }
  },
```

```
    async deleteRecord(ev) {
      try {
        await this.userStore.deleteUser(ev);
        this.$refs.child.cancel();
      } catch (error) {
        console.error(error);
      }
    },
  },
```

## Resultado de la anterior configuración