EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: PROPS.
- EXERCISE 2: PROPS AVANZADAS.
- EXERCISE 3: EVENTOS.

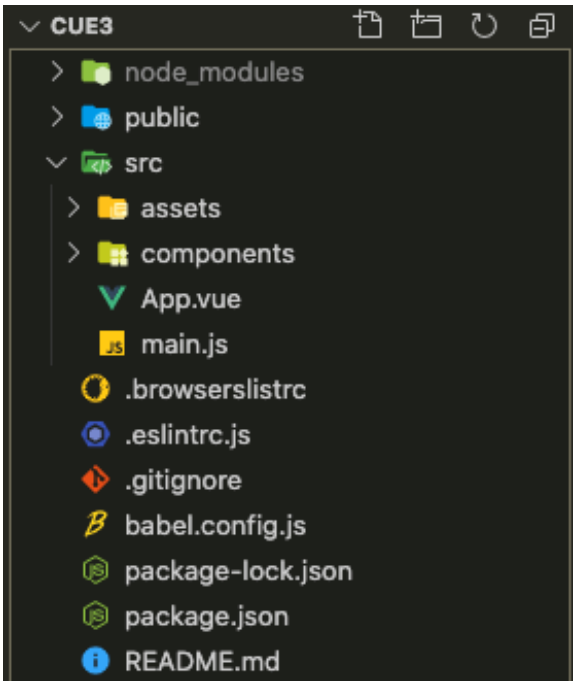
EXERCISE 1: PROPS.

Para iniciar, vamos a crear un nuevo proyecto; y para ello, haremos una carpeta llamada cue3, donde éste se desarrollará.

Seguiremos la misma guía utilizada en el Cue2: Ejercicio 2 (instalación vue/cli).

Y como nombre de proyecto, pondremos “cue3”.

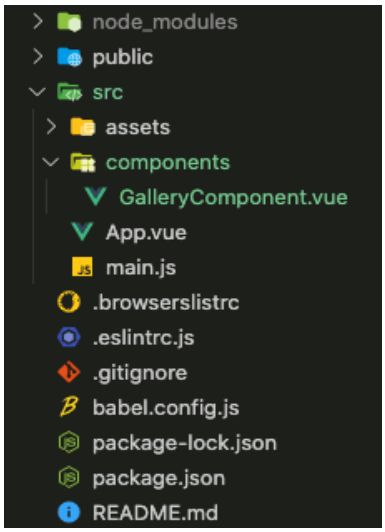
Una vez instalado, quitaremos el componente “HelloWorld.vue” de nuestra app, desde Visual Studio Code, y luego eliminaremos su referencia en App.vue, como lo hicimos en los ejercicios de cue2.



Nuestro fichero `App.vue`, quedará de la siguiente forma:

```
<template>
  <div id="app">
  </div>
</template>

<script>
export default {
  name: 'App',
  components: {

  }
}
</script>
<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

Para comenzar, crearemos un nuevo archivo en la carpeta `components`, llamado `"GalleryComponent.vue".`



Dentro del componente `GalleryComponent.vue`, en la sección de script, escribiremos el nombre del componente y el Array prop.

Props como array:

La forma más sencilla de poder crear props a un componente, es asignando un Array con las propiedades que se necesitan pasar. Esto lo haremos en la sección de script de dicho componente.

```
export default {
  name:"gallery-component",
  props:['title','images_src']
}
```

Una vez asignadas las 2 props, las aplicaremos en el template html.

Acceso a props dentro del componente:

Para poder acceder a los datos que están entrando como props al componente, lo hacemos de la misma forma que con datos declarados dentro de data.

```
<template>
  <div>
    <h1>{{title}}</h1>
  <img v-for="image in images_src" :key="image" :src="image"
alt="">
  </div>
</template>
```

1. La prop “title”, está siendo utilizada dentro de un H1.
2. La prop images_src, está dentro de un v-for porque es un Array.

El código final de nuestro componente GalleryComponent.vue , quedaría así:

```
<template>
  <div>
    <h1>{{title}}</h1>
 <img v-for="image in images_src" :key="image" :src="image"
alt="">
  </div>
</template>

<script>
export default {
  name:"gallery-component",
  props:['title','images_src']
}
</script>
<style scoped>
  div{
    border: 3px solid rgb(37,86,150);
    border-radius:10px;
    padding:10px;
    width:60%;
    margin:0 auto;
    background-color:rgb(58,136,226);
    color:white;
  }
  img{
    display:inline-block;
    border-radius:50%;
    margin-right: 10px;
    border:3px solid rgb(19,62,143);
  }
</style>
```

Una vez escrito el código, lo guardaremos e iremos al componente App.vue.

Desde componente padre:

En este caso, invocaremos el componente GalleryComponent.vue, y dentro de App.vue, el archivo principal.

```vue
<template>
  <div id="app">
    <GalleryComponent
      title="Mi Galería"
      :images_src="images">
    </GalleryComponent>
  </div>
</template>

<script>

import GalleryComponent from '@/components/GalleryComponent.vue'
export default {
  name: 'App',
  components: {
    GalleryComponent,
  },
  data: function(){
    return{
      images: [
        "http://lorempixel.com/200/200/sports",
        "http://lorempixel.com/200/200"
      ],
    }
  },
}
</script>
```

Una vez guardado el archivo, levantaremos nuestra app, escribiendo “npm run serve” desde el terminal de Visual Studio.

Ya levantada nuestra aplicación, iremos al navegador, y el resultado que debemos obtener se visualizaría así:

EXERCISE 2: PROPS AVANZADAS.

Para comenzar, seguiremos en el mismo componente GalleryComponent.vue. Esta vez aprenderemos una forma más precisa de declarar props.

Si bien pasar props a través de un Array es la manera más sencilla, en algunos casos necesitaremos mayor restricción, por ejemplo:

- Definir el tipo de datos de la prop.
- Definir si es requerida para el funcionamiento del componente.
- Definir un valor por default.

En cualquiera de estos casos, a través del Array no es posible, por lo que debemos definir la prop como un objeto.

Definiendo el tipo de dato por props:

Si deseamos definir el tipo de dato por cada prop, deberíamos definir un objeto de la siguiente forma:

```
<script>
export default {
  props:{
    title:String,
    images_src:Array
  }
}
</script>
```

Definiendo Props de manera avanzada:

Si definimos cada elemento de nuestra prop como un objeto, podemos ser más específicos, determinando qué tipo de dato tendrá la prop, o si esa props es requerida o no.

Esto nos ayuda a estructurar nuestro componente, y a la vez, si una props no es del tipo que hemos declarado, recibiremos un error en la consola del navegador, indicando que ésta no pertenece a lo definido.

Para realizar el ejercicio, reescribiremos las props de nuestro componente "GalleryComponent.vue", quedando de la siguiente manera:

```
<script>
export default {
  props:{
    title:{
      type:String,
      default:"Mi titulo por default"
    },
    images_src:{
      type: Array,
      required:true,
    }
  },
}
</script>
```

Si bien son los mismos datos, ahora están más específicos. Por ejemplo: en la prop "titulo", hemos asignado un valor default, lo que significa que de no venir esa props como entrada, el componente obtendrá el valor desde allí.

Por otra parte, la prop "images_src", aparte de definir que será un Array, indica que ésta es requerida para el funcionamiento del componente, y de no ir, generará un error por consola.

En este caso, el componente "GalleryComponent.vue", quedaría:

```
<template>
  <div>
    <h1>{{title}}</h1>
 <img v-for="image in images_src" :key="image" :src="image"
alt="">
  </div>
</template>

<script>
export default {
  props:{
    title:{
      type:String,
      default:"Mi titulo por default"
    },
    images_src:{
      type: Array,
```

```
      required:true,
    }
  },
}
</script>

<style scoped>
  div{
    border: 3px solid rgb(37,86,150);
    border-radius:10px;
    padding:10px;
    width:60%;
    margin:0 auto;
    background-color:rgb(58,136,226);
    color:white;
  }
  img{
    display:inline-block;
    border-radius:50%;
    margin-right: 10px;
    border:3px solid rgb(19,62,143);
  }
</style>
```

Si guardamos, notaremos que obtendremos el mismo resultado.

EXERCISE 3: EVENTOS.

Algunas veces, es necesario comunicar ciertos eventos generados por el usuario hacia un componente padre, en este caso, para ejecutar alguna acción.

En el siguiente ejercicio trabajaremos en el mismo componente “GalleryComponent.vue”. Ahora, agregaremos eventos a nuestro componente, para que el padre que lo esté renderizando genere acciones.

Eventos:

Para gatillarlos, se debe utilizar la palabra reservada $emit, la cual nos permite generar eventos que pueden ser escuchados por un componente padre.

¿Por qué eventos?

Al tener la opción de ir dividiendo nuestra aplicación en pequeñas partes (COMPONENTES), es necesario comunicarnos con el componente padre, para que éste genere alguna acción.

Ejemplo:

Abriremos el Proyecto en el cual hemos trabajo en este CUE, e iremos a “GalleryComponent”. En esta fase, es recomendable hacer un commit, para guardar los cambios del ejercicio 2.

Debajo de la etiqueta título, crearemos un formulario, el cual tendrá un label, un input, y un botón.

El input está ligado a el dato “newImage”, por lo que debemos crear el objeto data, e incluirlo.

El botón está ligado al método “agregar”.

```
<template>
  <div class="container">
    <h1>{{title}}</h1>
    <form>
      <label for="">Ingrese src imagen</label>
      <input type="text" v-model="newImage">
      <button class="btn btn-add"
@click.prevent="agregar">Agregar</button>
    </form>
    <div class="card" v-for="(image,index) in images_src"
:key="image" >
      <img :src="image" alt="img">
      <button class="btn btn-delete"
@click="eliminar(index)">Eliminar</button>
    </div>
  </div>
</template>
```

En el siguiente ejemplo hay 2 botones, los cuales están asociados a métodos. Si bien los eventos pueden ser gatillados desde el mismo template, en este caso, lo haremos más ordenado y será asociado a un método.

```
methods:{
    eliminar(index){
      this.$emit('delete',index);
    },
    agregar(){
      this.$emit('add',this.newImage);
      this.newImage = "";
    }
  }
```

Si observamos a detalle, los dos eventos tienen un dato asociado, el cual podrá ser leído por el padre cuando el evento se gatille.

El componente anterior “GalleryComponent.vue”, quedará de esta forma.

```
<template>
  <div class="container">
    <h1>{{title}}</h1>
    <form>
      <label for="">Ingrese src imagen</label>
      <input type="text" v-model="newImage">
      <button class="btn btn-add"
@click.prevent="agregar">Agregar</button>
    </form>
    <div class="card" v-for="(image,index) in images_src"
:key="image" >
      <img :src="image" alt="img">
      <button class="btn btn-delete"
@click="eliminar(index)">Eliminar</button>
    </div>
  </div>
</template>
<script>
export default {
  props:{
    title:{
      type:String,
      default:"Mi titulo por default"
    },
    images_src:{
```

```
      type: Array,
      required:true,
    }
  },
  data:function(){
    return{
      newImage:"",
    }
  },
  methods:{
    eliminar(index){
      this.$emit('delete',index);
    },
    agregar(){
      this.$emit('add',this.newImage);
      this.newImage = "";
    }
  }
}
</script>
<style scoped>
  .container{
    border: 3px solid rgb(37,86,150);
    border-radius:10px;
    padding:10px;
    width:60%;
    margin:0 auto;
    background-color:rgb(58,136,226);
    color:white;
  }
  img{
    display:block;
    border-radius:50%;
    margin-right: 10px;
    border:3px solid rgb(19,62,143);
  }
  .card{
    display:inline-block;
  }
  form{
    padding: 20px;
  }
  input{
    width:40%;
    height:25px;
  }
  .btn-delete{
    background-color: rgb(190, 28, 28);
  }
  .btn-add{
    background-color: rgb(28, 190, 63);
    margin-left:10px;
```

```
  }

  .btn{
    padding:10px;
    color:rgb(235, 231, 231);
    border:none;
    border-radius: 10px;
    width:100px;
    margin-top:10px;
    text-decoration: none;
    font-weight: 800;
  }
</style>
```

Componente Padre:

Ahora, iremos al componente App.vue, y agregaremos los eventos “delete” y “add, que serán asociados a métodos que crearemos.

```
<template>
  <div id="app">
    <GalleryComponent
      title="Mi Galería"
      :images_src="images"
      v-on:delete="eliminarImagen"
      @add="agregarImagen"
    ></GalleryComponent>
  </div>
</template>
```

Si te fijas, el evento delete es llamado con la directiva v-on, y “add” es llamado con @. Esto es para ejemplificar que @ es el atajo para v-on.

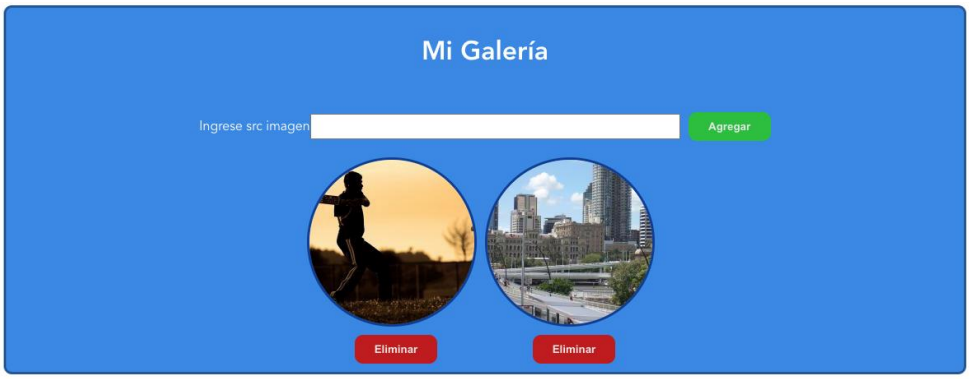
```

<script>
import GalleryComponent from '@/components/GalleryComponent.vue'
export default {
  name: 'App',
  components: {
    GalleryComponent,
  },
  data: function(){
    return{
```

```
      images:
["http://lorempixel.com/200/200/sports","http://lorempixel.com/200/200"],
    }
  },
  methods:{
    eliminarImagen(index){
      this.images.splice(index,1);
    },
    agregarImagen(src){
      this.images.push(src);
    }
  }
}
</script>
```

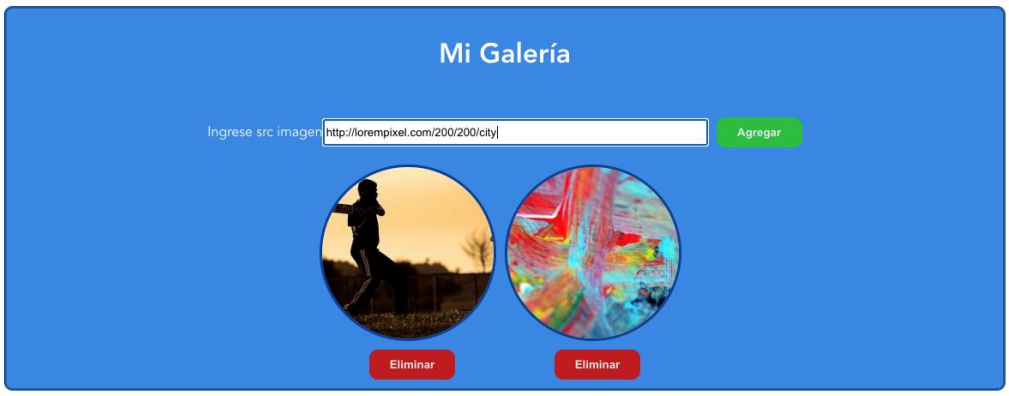
El componente hijo gatilla un evento, y el padre es el encargado de ejecutar la acción. En el caso del ejemplo, ésta es borrar o agregar un elemento a un Array.

Ahora, si guardamos y levantamos nuestra app con “npm run serve”, veremos lo siguiente:

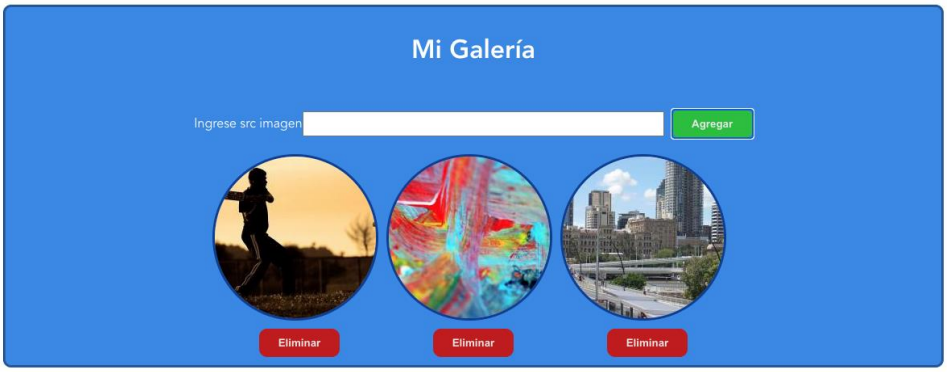
Aquí podremos agregar o eliminar imágenes a nuestra galería.



Si agregamos la imagen: http://lorempixel.com/200/200/city



Al presionar “agregar una tercera imagen”, ésta será incluida en nuestra galería; y si presionamos “eliminar” en alguna de ellas, serán borradas. En el ejemplo, eliminaremos la segunda imagen.



Al presionar “Eliminar” en la segunda imagen, el resultado es el siguiente: