

Agregando botones y funcionalidades para Crear, editar y eliminar

Básicamente todo lo haremos desde el componente Vue. Seguimos estos pasos:

Para Eliminar registros desde la Vista Principal

1. Creamos el respectivo encabezado:

```
<th scope="col">Teléfono</th>
<th scope="col">Dirección</th>
<th scope="col" colspan="2" class="text-center">Acción</th>
</tr>
</thead>
```

Como se ve, con colspan se agrupan dos columnas en una

2. En las filas donde van los registros, agregamos dos botones: Editar y eliminar

```
<td>{{ contacto.tele }}</td>
<td>{{ contacto.direccion }}</td>
<td>
  <button class="btn btn-warning">Editar</button>
</td>
<td>
  <button class="btn btn-danger">Eliminar</button>
</td>
</tr>
</tbody>
```

3. Agregamos un nuevo método o función en el script del mismo componente:

```
this.contactazos = respuesta.data //en todos los casos se debe poner .data
},
async eliminar(id) {
  const respuesta = await axios.delete('/contactos/' + id) //un await solo
  //de un async
  this.listar() //con esto no tendremos que estar actualizando el navegador
}
},
created() {
  this.listar() //con created() se carga la función o los datos tan pronto
```

Como puede apreciarse, en vez del método get, se usa el método delete. Además, se agregó una barra inclinada al principio de la dirección o ruta requerida y otra más al final a la cual también se le concatena el id. Este id coincide con el id de la tabla Contactos, y que luego también se especificará dentro del botón mismo de eliminar en el template.

Asimismo, se llama al método listar dentro del método eliminar, esto con el fin de reconstruir en pantalla la información del array actualizado y no tener que estar actualizando el navegador cada que se elimine un registro.

- Se hace el llamado respectivo a la función eliminar, desde el template, en el botón eliminar, con @click, así:

```
</td>
<td>
  <button @click="eliminar(contacto.id)" class="btn btn-danger">Eliminar</button>
</td>
</tr>
```

- Se hace la codificación necesario en el método destroy() del api controller (recordar que se llama ContactController):

```
app > Http > Controllers > ContactController.php > ContactContro
61 public function destroy(Contact $contacto)
62 {
63     $contacto->delete();
64 }
65 }
```

Por default, aparece la instancia \$contact pero en algunos casos, esto da un error inesperado. Por eso, se recomienda colocarlo en español o con otro nombre.

- Probamos y veremos que ya podemos borrar desde el navegador. En la captura, se eliminaron los registros 3 y 4

Id	Nombre	Apellido	Email	Teléfono	Dirección	Acción	
1	Maddison Cole	Gutkowski	timmy.kozey@example.com	987654321	17481 Tromp Hills Metzside, RI 95268	Editar	Eliminar
2	Matilde Kemmer	Turcotte	cbeier@example.net	123456789	636 Cristopher Coves Apt. 236 West Freddiebury, NJ 90354	Editar	Eliminar
5	Sherman Goldner	Reinger	robb.kozey@example.net	987654321	1263 Dare Greens Apt. 351 West Shea, NV 24172-8377	Editar	Eliminar
6	Dr. Otto Sanford	Beier	thompson.mohammad@example.com	987654321	841 Lola Parkways Apt. 689 New Gregory,	Editar	Eliminar

Implementando el botón para crear Nuevo Contacto.

Sigamos estos pasos:

- Vamos a Bootstrap y buscamos Modal. Link <https://getbootstrap.com/docs/5.2/components/modal/#how-it-works>. Allí bajamos por la página y ubicamos Live Demo

Live demo

Toggle a working modal demo by clicking the button below. It will slide down and fade in from the top of the page.

Launch demo modal

```
<!-- Button trigger modal -->
<button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#exampleModal">
  Launch demo modal
</button>

<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModallabel" aria-hi
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModallabel">Modal title</h5>
```

2. Copiamos ese Código.
3. Vamos a nuestro componente y pegamos debajo de la etiqueta hr:
4. Quitamos del botón la parte de data-bs-toggle y data-bs-target.

```
<hr>
<!-- Added Button trigger modal -->
<button type="button" class="btn btn-primary">
  Nuevo contacto
</button>
```

```
<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModallabel" aria-hidden="true"
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModallabel">{{titleModal}}</h5>
```

5. También quitamos todo lo que está encerrado con la línea naranja, de tal modo que quede así:

```
<!-- Modal -->
<div class="modal">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModallabel">{{titleModal}}</h5>
```

6. Le añadimos una clase dinámica (con :class, algo propio de Vue), de tal modo que el estilo que vamos a crear más abajo, solo se active si se cumple una condición. Observe la siguiente captura:

```
<div class="modal" :class="{mostrar:modal}"> <!--Esto es una clase dinámica. De hecho, mostrar es una clase
que creamos en la sección style. Solo se activará si el atributo modal es true,
<div class="modal-dialog">
  <div class="modal-content">
    <div class="modal-header">
```

Como puede verse en el mismo comentario del código, existe una clase llamada **‘mostrar’** cuyo estilo CSS solo se activará si se cumple una condición, la cual se expresa con dos puntos. En este caso la condición se llama **modal**, que quiere decir que si modal es **true**, la clase **‘mostrar’** se activará. Para que esto funcione, modal debe ser un atributo con un valor que cambia dinámicamente. Si recordamos un poco de programación básica, una variable o atributo puede ser usado dentro de una comparación o condición como, por ejemplo: *modal == 10*, *modal > 20*, *modal <= 15*, etc. Si

solamente dice modal, es porque está preguntando si es verdadera, en este caso actúa como booleano.

Pero, ¿dónde definimos este atributo booleano? En el script del componente.

7. Digite lo siguiente dentro del script:

```
<script>
  export default {
    data() {
      return {
        modal: 0, /*si el modal está cerrado será 0 y si está abierto será 1. Por default
                  lo inicializamos en 0 que significa false */
      }
    }
  }

```

Esta variable se inicializa en cero (0) que significa false.

8. También aprovechamos y agregamos el atributo tituloModal, que se inicializa como vacío:

```
data() {
  return {
    modal: 0, /*si el modal está cerrado
              lo inicializamos en 0 que
    tituloModal: '',
    contactazos: [], //no importa cómo bu
  }
}
```

9. Pero si se supone que modal va a cambiar entre 0 y 1 (false y true), ¿cómo se logra eso ya que en este momento es estático? Vamos por partes. Por ahora, dentro de methods, donde ya tenemos los métodos listar y eliminar, creamos dos métodos nuevos:

```
async eliminar(id) {
  const respuesta = await axios.delete(
    this.listar() //con esto no tendremos
  },
  openModal() {
    this.modal = 1
  },
  closeModal(){
    this.modal = 0
  }
}
```

10. Ya hablamos de una clase 'mostrar' cuyos estilos se activarán si se cumple la condición de modal como true (1). Es necesario crear esa clase dentro de la sección de <style>. Digite:

```
02 <style>
03   .mostrar { /*Solo se activará si modal es 1 (true) */
04     display: list-item;
05     opacity: 1;
06     background: rgba(44, 38, 75, 0.849);
07   }
08 </style>
```

11. Dentro del código del modal de Bootstrap que ya pegamos en nuestro componente (punto 3), debemos implementar el atributo tituloModal del punto 8 y los métodos del punto 9.

Con @click (que es lo mismo que v-on:click), estamos indicando que si se da clic en el botón se activen las funciones respectivas. Por ejemplo, si se da clic en el botón principal Nuevo contacto, se desencadena las rutinas del método openModal(). Es decir, el atributo modal pasará a valer 1 (es decir true) y, por ende, se activará la case 'mostrar'. Igualmente, hacemos los demás cambios indicados en la captura:

```
<button @click="openModal()" type="button" class="btn btn-primary">
  Nuevo contacto
</button>
```

```
<div class="modal-dialog">
  <div class="modal-content">
    <div class="modal-header">
      <h5 class="modal-title" id="exampleModalLabel">{{tituloModal}}</h5>
      <button @click="closeModal()" type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
    </div>
    <div class="modal-body">
      ...
    </div>
    <div class="modal-footer">
      <button @click="closeModal()" type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cerrar</button>
      <button type="button" class="btn btn-success">Guardar cambios</button>
    </div>
  </div>
```

- Para poder implementar título para el modal, creamos el atributo update en el script, inicializándolo en true:

```
export default {
  data() {
    return {
      update: true,
      modal: 0, /*si el modal
               lo iniciamos en 0*/
    }
  }
}
```

- Luego para el método openModal() implementamos la siguiente serie de condiciones. Si update es true (que así de hecho se inicializó), entonces el título del modal será "Modificar contacto":

```
openModal() {
  this.modal = 1
  if(this.update){
    this.tituloModal = "Modificar contacto"
  }else{
    this.tituloModal = "Crear contacto"
  }
}
```

- Para el botón de "Nuevo contacto" indicamos que update es false en caso de que se dé clic en el botón. Como ya tenemos la directiva @click, en ella misma lo agregamos así:

```
<button @click="update=false; openModal()" type="button">
  Nuevo contacto
</button>
```

15. Así como está declarada esta directiva, la podemos declarar en el botón Editar. Pero cambiamos el valor de update por 'true'. Además, para hacer la edición, requerimos el id, tal como lo hicimos para eliminar. Por eso lo llamamos dentro del método openModal como 'contact.id':

```
<td>{{ contacto.direccion }}</td>
<td>
  <button @click="update=true; openModal(contacto.id)" class="btn btn-warning">Editar</button>
</td>
```

16. Por lo tanto, en el método openModal, en nuestro script, será necesario también indicar que se requiere el id. En ese caso, lo inicializamos en cero. Es decir, si no se pasa ningún parámetro, ese es su valor por default:

```
openModal(id=0) {
  this.modal = 1
  if(this.update){
    this.tituloModal = "Modificar contacto"
  }else{
    this.tituloModal = "Agregar contacto"
  }
}
```

Guardar datos de nuevo contacto

1. Empecemos por crear un objeto en nuestro script, al que llamaremos contacto y tendrá unos datos de prueba:

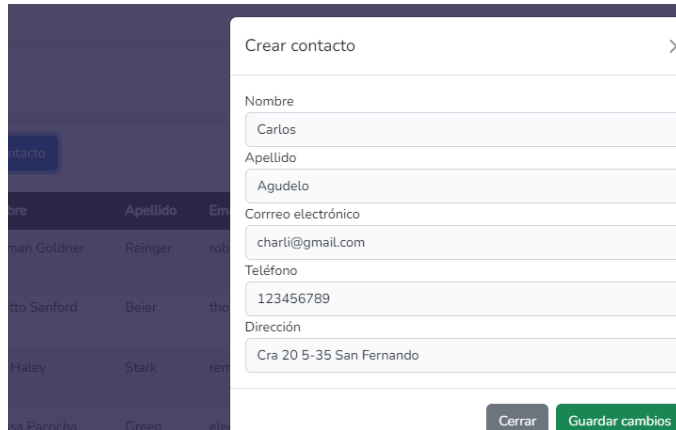
```
export default {
  data() {
    return {
      contacto: {
        nombre: 'Carlos',
        apellido: 'Agudelo',
        email: 'charli@gmail.com',
        tele: 123456789,
        direccion: 'Cra 20 5-35 San Fernando'
      },
      update: true,
      modal: 0, /*si el modal está cerrado será 0 y si está abierto será 1
                 lo inicializamos en 0 que significa falso*/
      tituloModal: '',
      contactazos: [], //no importa cómo bauticemos el array
    }
  }
}
```

2. Conectamos cada input del modal con este objeto. Para eso, se usa la propiedad reactiva v-model, así:

```
<div class="modal-body">
  <div class="form-group">
    <label for="nombre">Nombre</label>
    <input v-model="contacto.nombre" type="text" name="" id="" />
  </div>
  <div class="form-group">
    <label for="apellido">Apellido</label>
    <input v-model="contacto.apellido" type="text" name="" id="" />
  </div>
  <div class="form-group">
```

Hacemos algo similar con los demás inputs.

- Guardamos, y cuando vamos a la página web, y abrimos el modal, salen nuestros datos de prueba que le pasamos al objeto `contacto`:



- Ahora, no queremos que nos muestre unos datos estáticos, sino de manera reactiva. Si vamos a crear un usuario nuevo, los inputs del modal deberían estar vacíos, y si vamos a editar, los inputs deberían mostrar los datos del usuario seleccionado. Para esto, seguimos estos pasos:

- Quitamos los datos de prueba de los campos del objeto `contacto`, y dejamos estos últimos inicializados como vacíos:

```
data() {  
  return {  
    contacto: {  
      nombre: '',  
      apellido: '',  
      email: '',  
      tele: null,  
      direccion: ''  
    },  
    update: true,  
  }  
}
```

- En el método `openModal()` del script, cambiamos el parámetro por `datos = {}`, lo que significa que estamos inicializando un objeto como totalmente vacío, y le hemos llamado `datos`:

```
openModal(datos = {}) {
```

- Para el caso de editar, que se activa si se cumple la condición de `update == true`, hacemos que el objeto **`contacto`** se rellene con lo que venga por default desde la tabla `contactos` de la base de datos, es decir, **`contacto = datos`**. Debe quedar así:

```
openModal(datos = {}) {  
  this.modal = 1  
  if(this.update){  
    this.contacto = datos  
    this.tituloModal = "Modificar contacto"  
  }  
}
```

- 4.4. Para el caso de crear (es decir `update == false`), retornamos los campos tal cual como están en el objeto **contacto**, es decir, vacíos. Para eso, digitamos lo siguiente (solo lo encerrado por las líneas naranja):

```
if(this.update){
  this.contacto = datos
  this.tituloModal = "Modificar contacto"
}else{
  this.tituloModal = "Crear contacto"
  return this.contacto
}
```

- 4.5. Hacemos el siguiente cambio en la línea para el botón Editar en el modal, en el template:

```
<button @click="update=true; openModal(contacto)" class="btn btn-warning">Editar</button>
```

Esto es para que ya no tenga como parámetro solamente el id, sino todos los datos del array contacto. No confundir este contacto (el cual es el alias del array `contactazos` de nuestro `foreach`), con el objeto `contacto` del script.

5. Para que pueda guardar los datos lo haremos así:

- 5.1. Creamos una nueva función asíncrona, llamada `guardar()`. Si es un registro nuevo, con `axios` llamamos al método `post` y a la ruta `/contactos`, la cual de acuerdo al listado de rutas de Laravel, es la que necesitamos para usar el método `store()` del controlador:

```
async guardar(id){
  if(this.update){ //si el atributo update es true. Aplica para botón Editar
  } else { //si el atributo update es false. Aplica para botón Crear nuevo
    const respuesta = await axios.post('/contactos', this.contacto)
  }
}
```

- 5.2. Queremos que una vez guardado el contacto nuevo (o actualizado uno existente), el modal se cierre y que se muestre la lista actual de contactos. Para eso llamamos los respectivos métodos (ojo, después de cerrar el `else`):

```
} else { //si el atributo update es false. Aplica para botón Crear nuevo
  const respuesta = await axios.post('/contactos', this.contacto)
}
this.closeModal()
this.listar()
},
```

- 5.3. Enlazamos el botón de Guardar cambios, en el modal, con el método que acabamos de crear, así:


```

<div class="modal-footer">
  <button @click="closeModal()" type="button" class="btn btn-secondary" data-bs-
  <button @click="guardar()" type="button" class="btn btn-success">Guardar camb
</div>
</div>

```

5.4. Nos aseguramos que en el controlador ContactController, el método store() tenga la rutina apropiada:

```

public function store(Request $request)
{
    $contacto = new Contact();
    $contacto->create($request->all());
}

```

Es de tener en cuenta, que si existe un input de tipo 'file', con esta rutina no se podrá subir un archivo como tal. En ese caso se puede probar el método `fill($request->except('miarchivo'))`. De todos modos, puede ser necesario investigar más al respecto, y cómo trata Vue los archivos, ya que desde Laravel sabemos perfectamente cómo se procesan.

5.5. Probamos creando un nuevo usuario. Luego clic en Guardar cambios:

Veremos el usuario creado al final de la tabla:

30	Mr. Maxine Baustien Jr.	neutel	garrecuassim@example.com	907054321	5522 Wilkinson Islands Havemouth, WV 41634-0651	Editar
31	Carlos Andres	Agudelo	charli@gmail.com	123456789	Cra 20 Barrio San Fernando	Editar