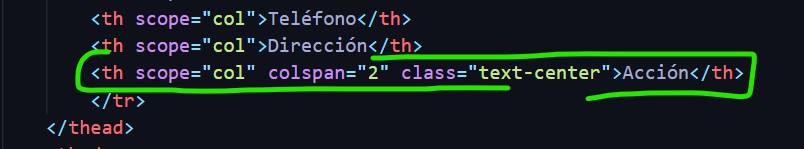
***Agregando botones y funcionalidades para Crear, editar y eliminar***

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

**Para Eliminar registros desde la Vista Principal**

1. Creamos el respectivo encabezado:



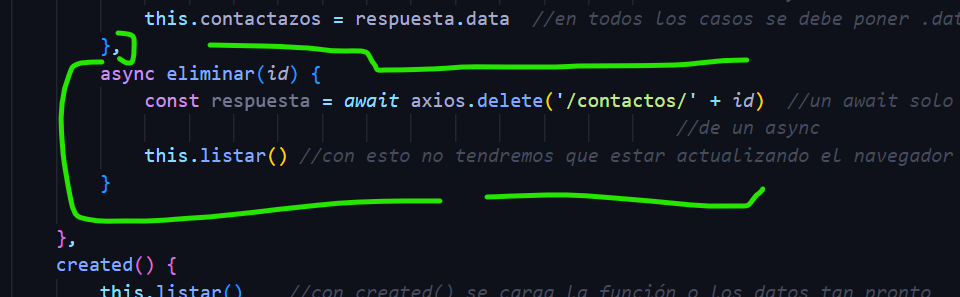
Como se ve, con colspan se agrupan dos columnas en una

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

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

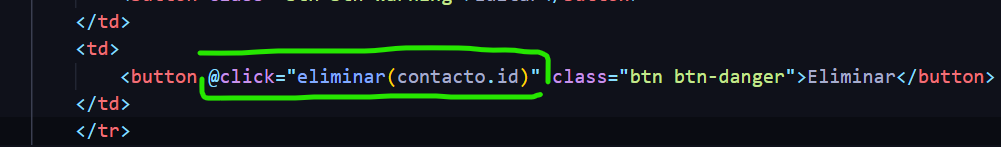
1. Agregamos un nuevo método o función en el scritp del mismo compontente:



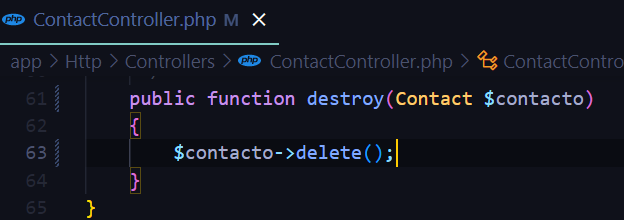
Como puede apreciarse, en vez del método get, se usa el método delete. Además, se agregó una barra inclinada al principio de la dirección o ruta requerida y otra más al final a la cual también se le concatena el id. Este id coincide con el id de la tabla Contactos, y que luego también se especificará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

1. Se hace el llamado respectivo a la función eliminar, desde el template, en el botón eliminar, con @clic, así:



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



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.

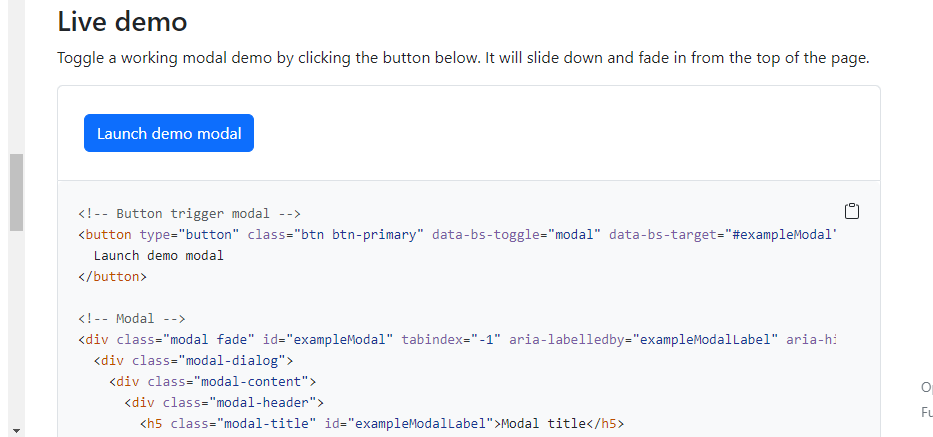
1. Probamos y veremos que ya podemos borrar desde el navegador. En la captura, se eliminaron los registros 3 y 4



**Implementando el botón para crear Nuevo Contacto.**

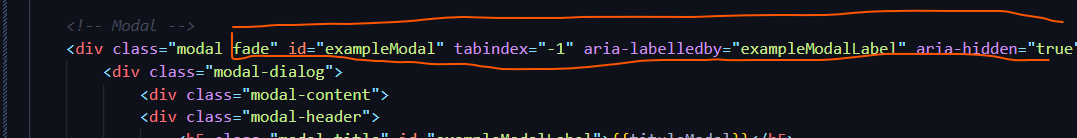
Sigamos estos pasos:

1. 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



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



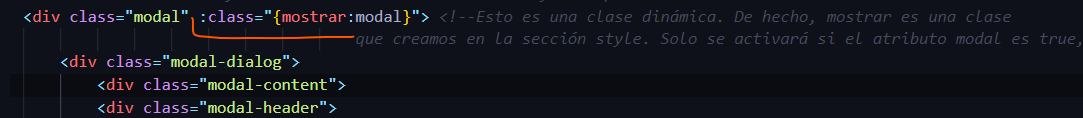


1. También quitamos todo lo que está encerrado con la línea naranja, de tal modo que quede así:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. 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:



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.

1. Digite lo siguiente dentro del script:

Texto

Descripción generada automáticamente

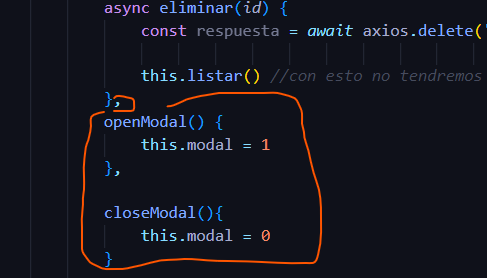
Esta variable se inicializa en cero (0) que significa false.

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

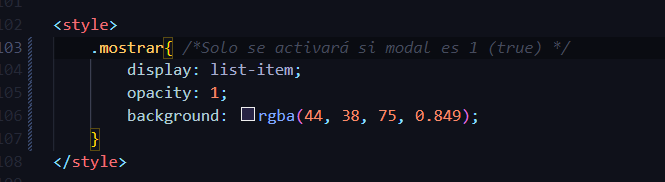
Texto

Descripción generada automáticamente

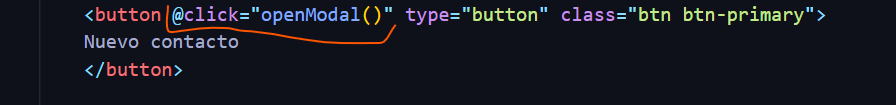
1. Pero si se supone que modal va a cambiar entre 0 y 1 (false y true), ¿cómo se logra eso ya que en este momento es estático? Vamos por partes. Por ahora, dentro de methods, donde ya tenemos los métodos listar y eliminar, creamos dos métodos nuevos:

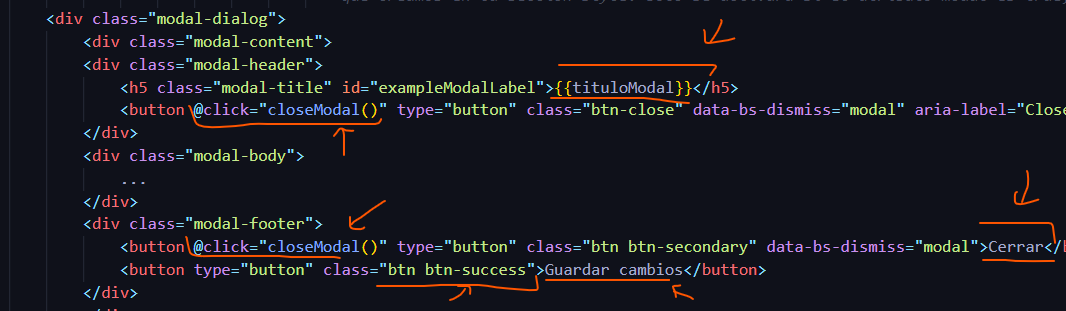


1. Ya hablamos de una clase ‘mostrar’ cuyos estilos se activarán si se cumple la condición de modal como true (1). Es necesario crear esa clase dentro de la sección de <style>. Digite:

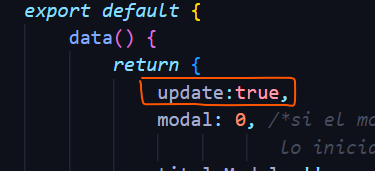


1. Dentro del código del modal de Bootstrap que ya pegamos en nuestro componente (punto 3), debemos implementar el atributo tituloModal del punto 8 y los métodos del punto 9. Con @click (que es lo mismo que v-on:click), estamos indicando que si se da clic en el botón se activen las funciones respectivas. 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:

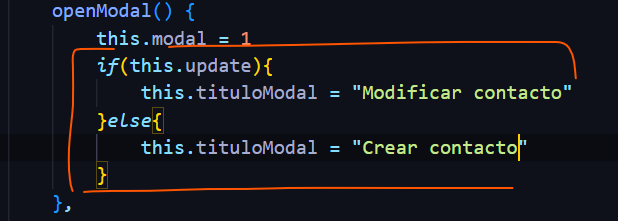




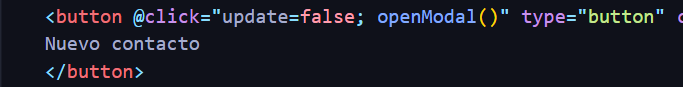
1. Para poder implementar título para el modal, creamos el atributo update en el script, inicializándolo en true:



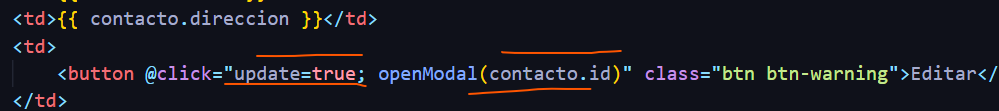
1. Luego para el método openModal() implementamos la siguiente serie de condiciones. Si update es true (que así de hecho se inicializó), entonces el título del modal será “Modificar contacto”:



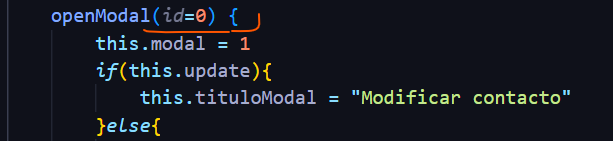
1. Para el botón de “Nuevo contacto” indicamos que update es false en caso de que se dé clic en el botón. Como ya tenemos la directiva @click, en ella misma lo agregamos así:



1. 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’:

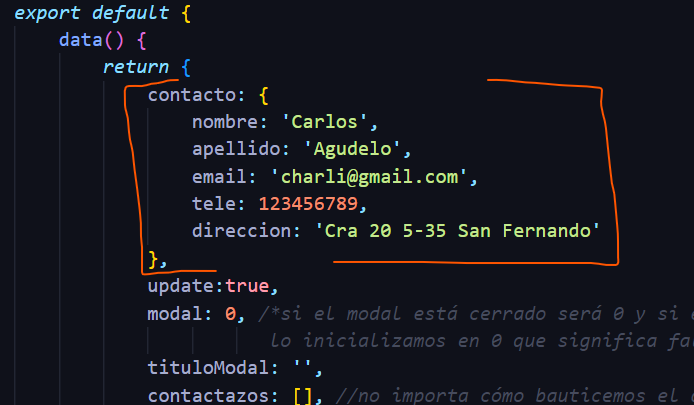


1. Por lo tanto, en el método openModal, en nuestro script, será necesario también indicar que se requiere el id. En ese caso, lo inicializamos en cero. Es decir, si no se pasa ningún parámetro, ese es su valor por default:

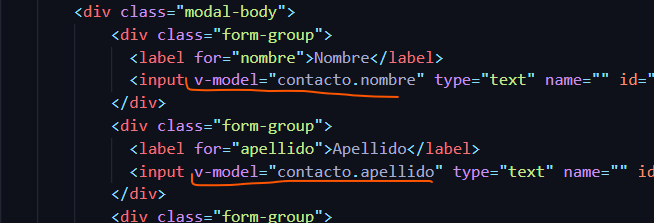


**Guardar datos de nuevo contacto**

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

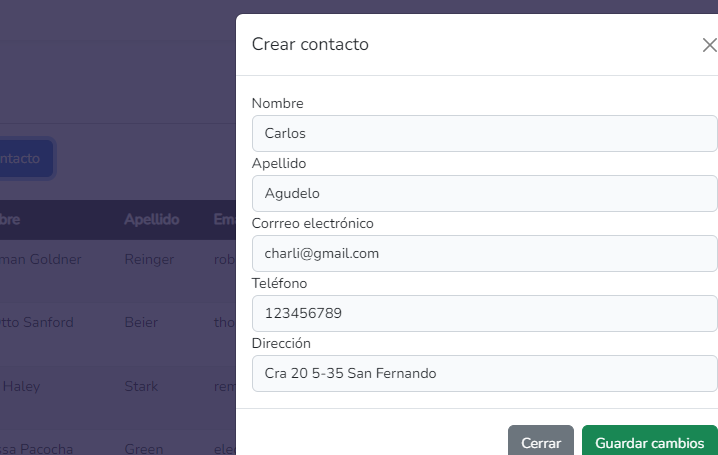


1. Conectamos cada input del modal con este objeto. Para eso, se usa la propiedad reactiva v-model, así:

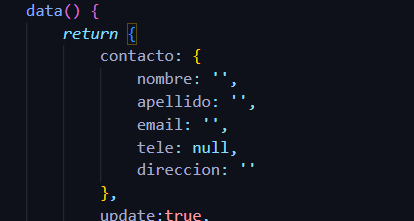


Hacemos algo similar con los demás inputs.

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



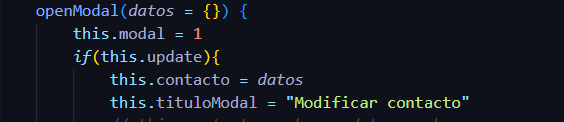
1. Ahora, no queremos que nos muestre unos datos estáticos, sino de manera reactiva. Si vamos a crear un usuario nuevo, los inputs del modal deberían estar vacíos, y si vamos a editar, los inputs deberían mostrar los datos del usuario seleccionado. Para esto, seguimos estos pasos:
   1. Quitamos los datos de prueba de los campos del objeto contacto, y dejamos estos últimos inicializados como vacíos:



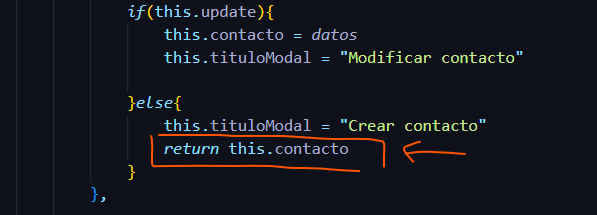
* 1. En el método openModal() del script, cambiamos el parámetro por datos = {}, lo que significa que estamos inicializando un objeto como totalmente vacío, y le hemos llamado datos:



* 1. 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:



* 1. 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):

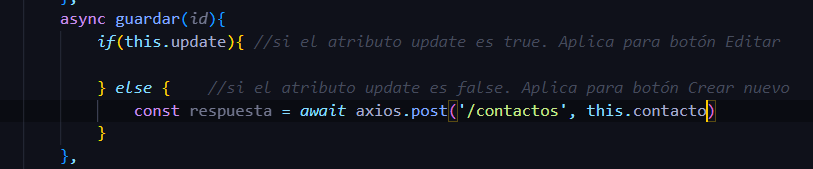


* 1. Hacemos el siguiente cambio en la línea para el botón Editar en el modal, en el template:

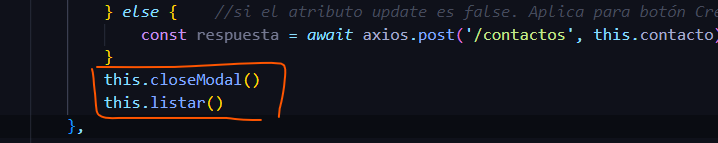


Esto es para que ya no tenga como parámetro solamente el id, sino todos los datos del array contacto. No confundir este contacto (el cual es el alias del array contactazos de nuestro foreach), con el objeto contacto del script.

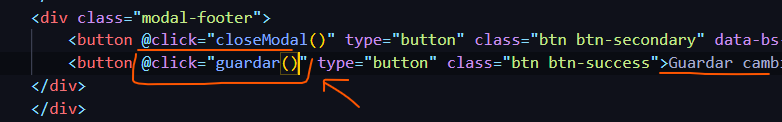
1. Para que pueda guardar los datos lo haremos así:
   1. Creamos una nueva función asíncrona, llamada guardar(). Si es un registro nuevo, con axios llamamos al método post y a la ruta /contactos, la cual de acuerdo al listado de rutas de Laravel, es la que necesitamos para usar el método store() del controlador:



* 1. 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):



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



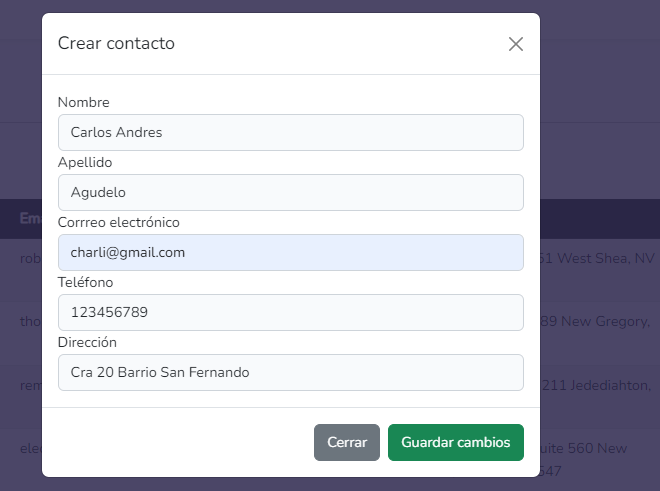
* 1. Nos aseguramos que en el controlador ContactController, el método store() tenga la rutina apropiada:

Texto

Descripción generada automáticamente con confianza media

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.

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



Veremos el usuario creado al final de la tabla:

