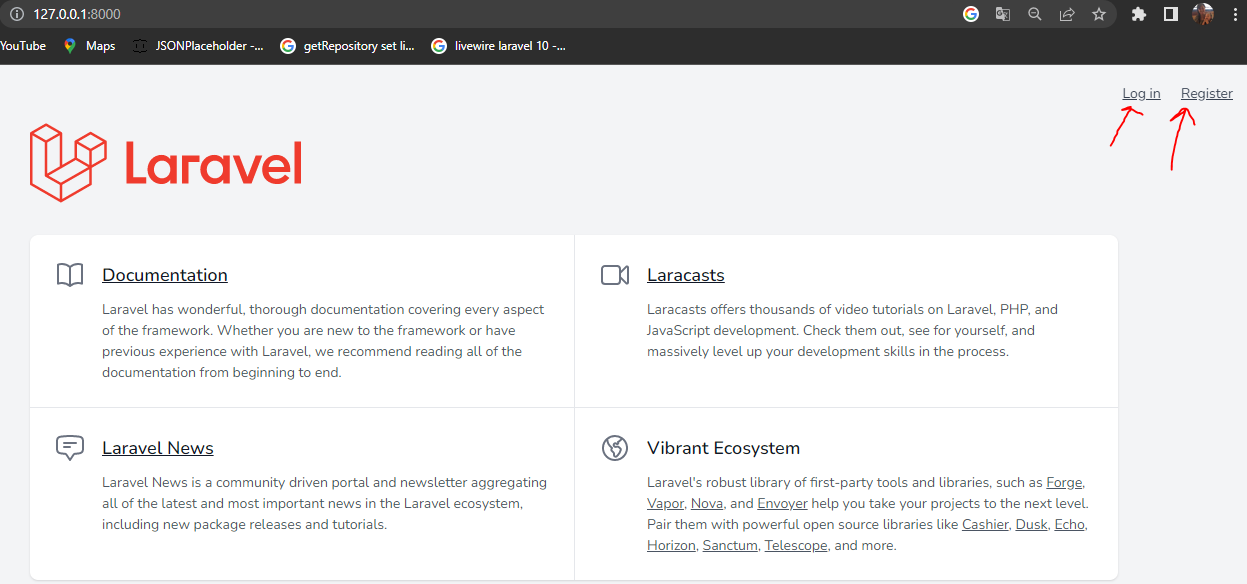
PRÁCTICA 3. CRUD EN LARAVEL 10 + JESTREAM + LIVEWIRE

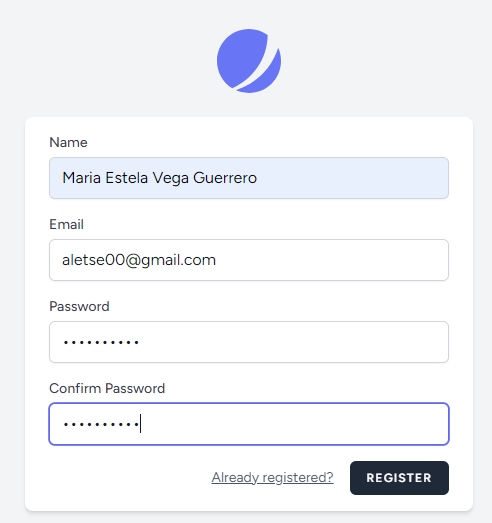
## 3.1 RECONOCIMIENTO DE LA INSTALACIÓN

### 3.1.1 Registro/login

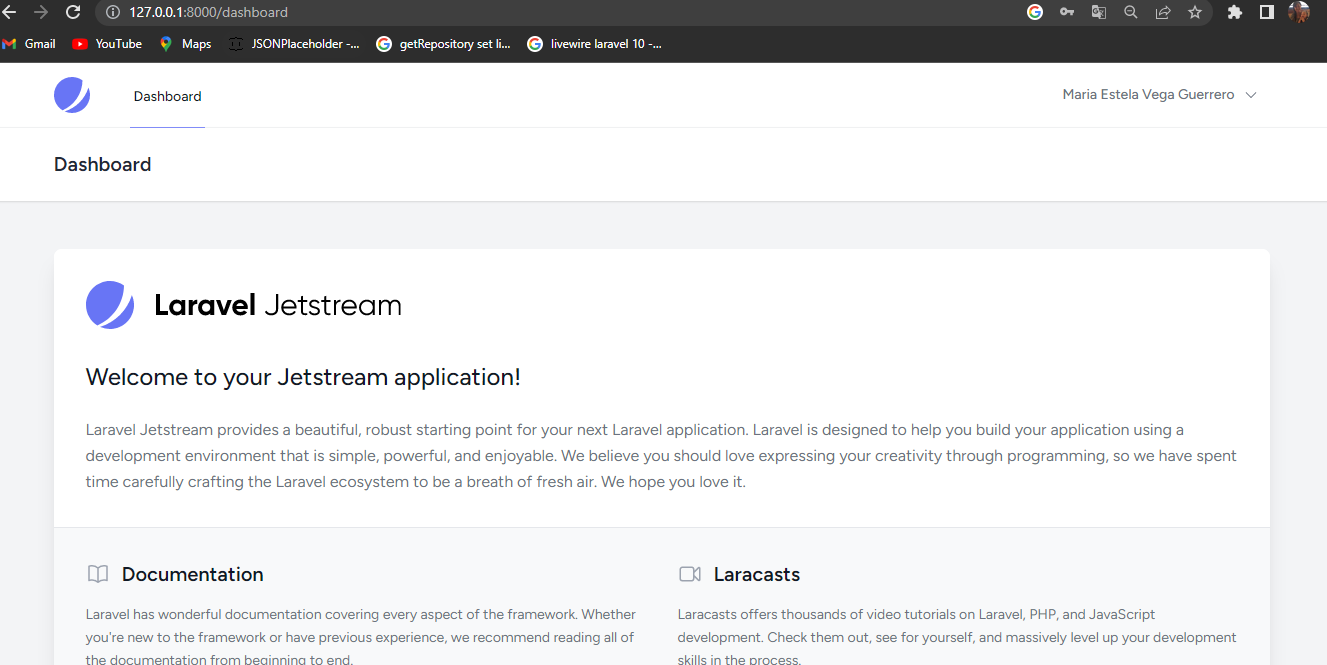
Cuando entramos por primera vez, nos da la opción de entrar a login o de registrar una cuenta.



Register



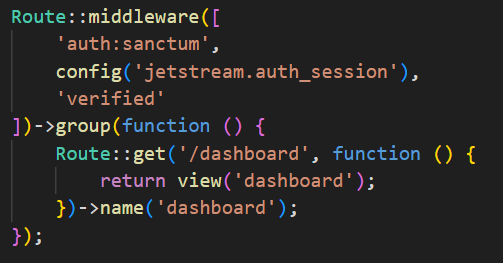
Una vez registrado entramos como autenticados



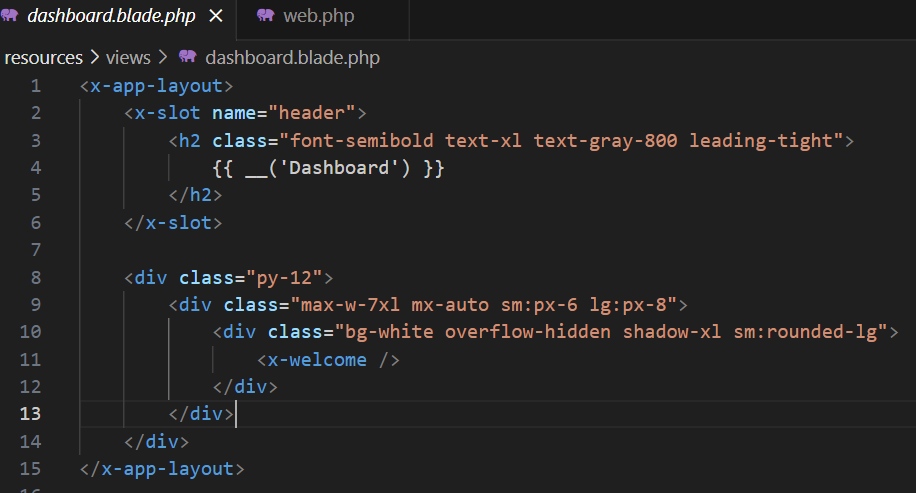
### 3.1.2 Programas básicos que se ejecutan

Iniciamos en donde ya se esta autenticado

Archivo web.php o rutas del sistema



Una vez logiado se ejecuta la ruta /*dashboard*, que llama a la vista *dashboard*



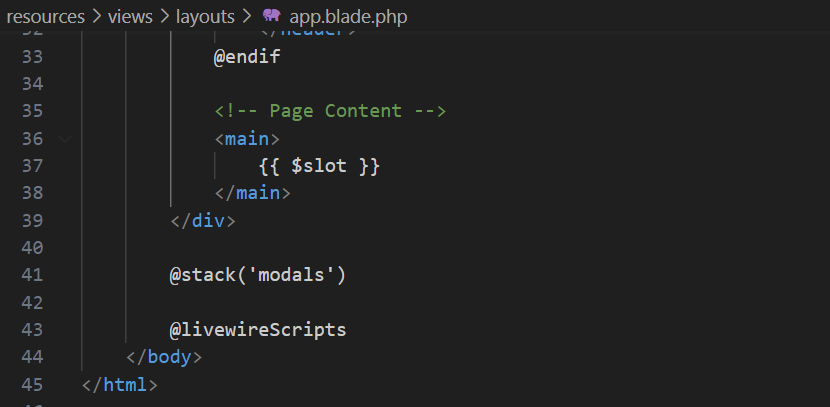
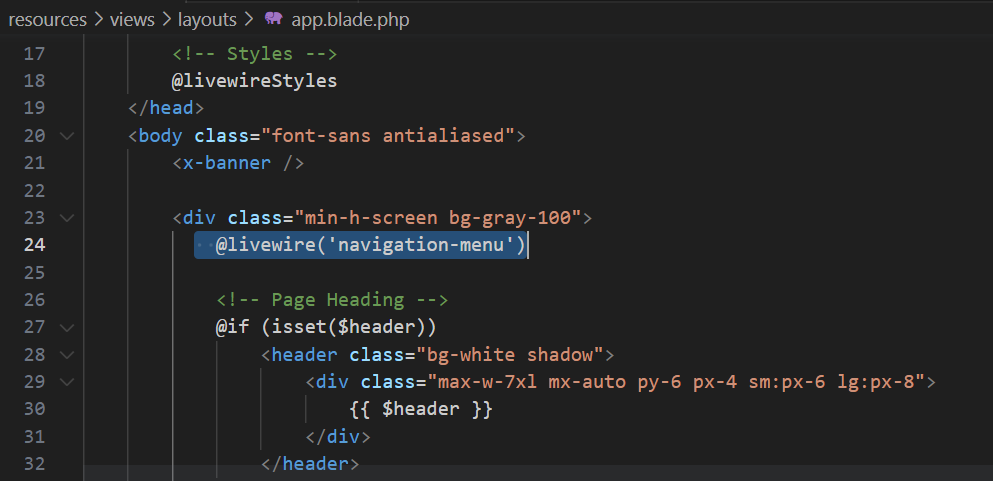
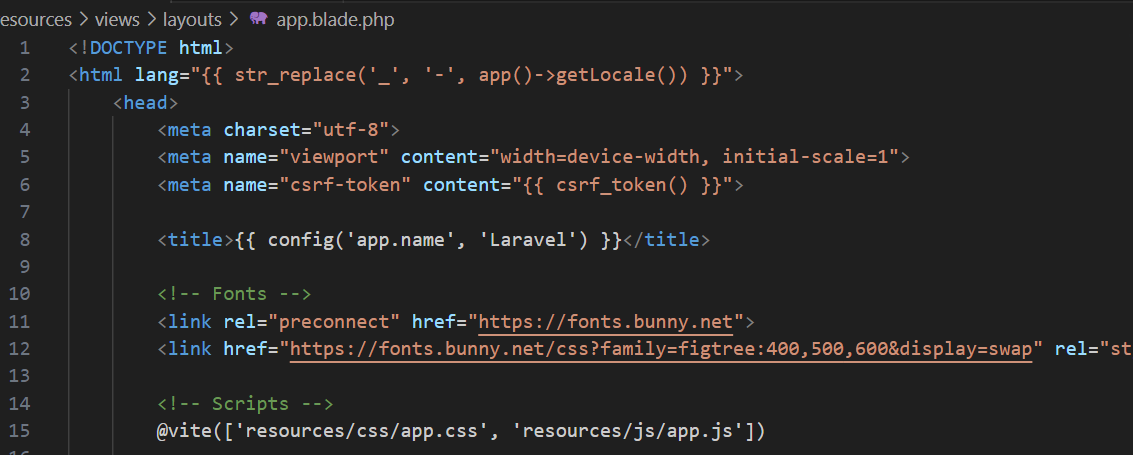
**Primero** se extiende de layouts/app.blade, por la línea 1 de la vista *dashboard*

**Segundo** separa el encabezado identificado en la línea 2 de *name=”header”*

**Tercer** separa la bienvenida con la agregación de un componente llamado *welcome*

En seguida se detallas las tres divisiones

**Primera**



En donde se estructura una página web completa desde que inicia (línea 1) hasta que finaliza (línea 45)

Define sus líneas de head, incluye metas, links, también una directiva relacionada con vite y una de livewirestyle

@vite(['resources/css/app.css', 'resources/js/app.js'])

Significa que todo componente de livewire será actualizado para su uso en esos dos archivos.

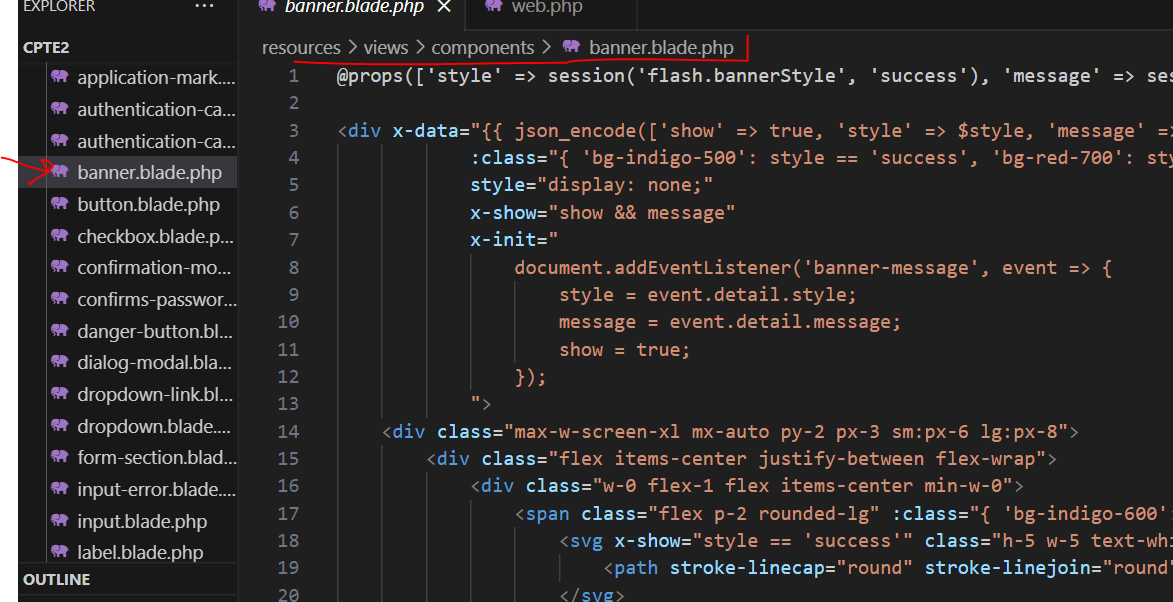
@livewireStyles

Significa que si se utilicen los estilos de los componentes (es una regla de livewire)

En la sección de body, incluye un banner especial que se incluye al instalar jeastrem

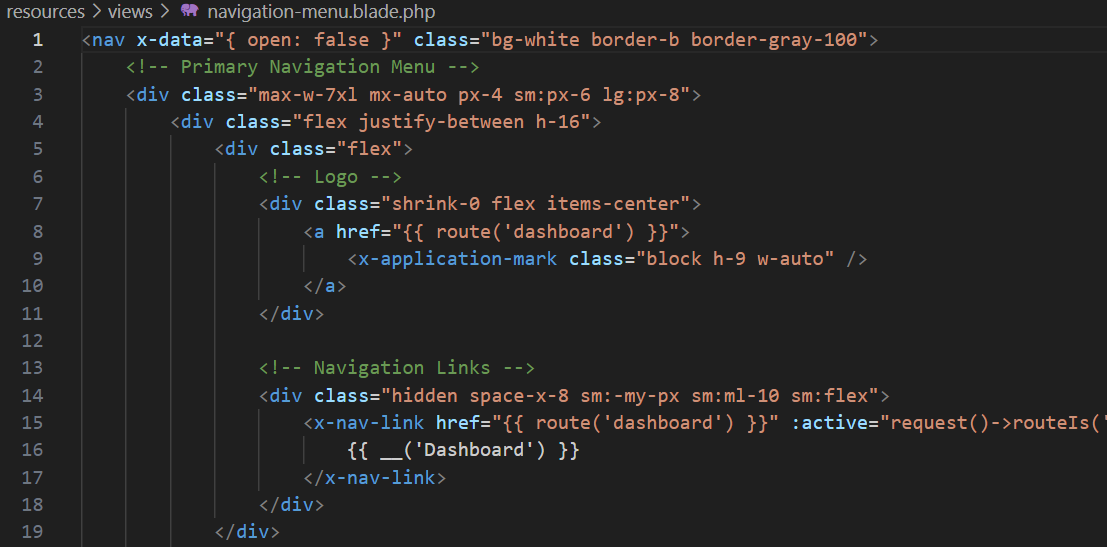
    <x-banner />

Componente:

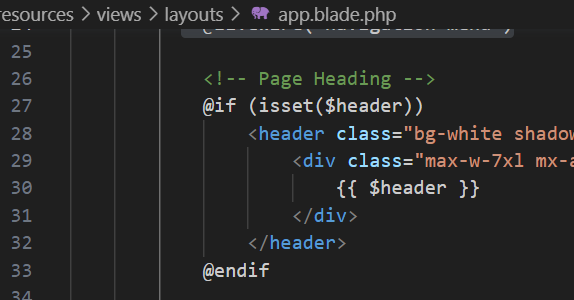


Continuamos en la sección body, se agrega un componente ya preparado de livewire llamado *navigation-menu*

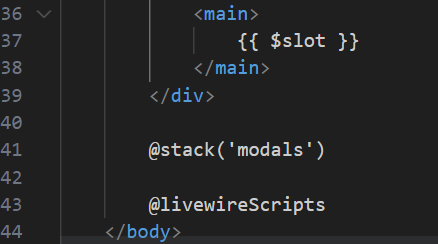
  @livewire('navigation-menu')



Continuando en la sección body, se incluye una pregunta donde si existe una variable llamada $heder se agregue en la sección línea 28 -32

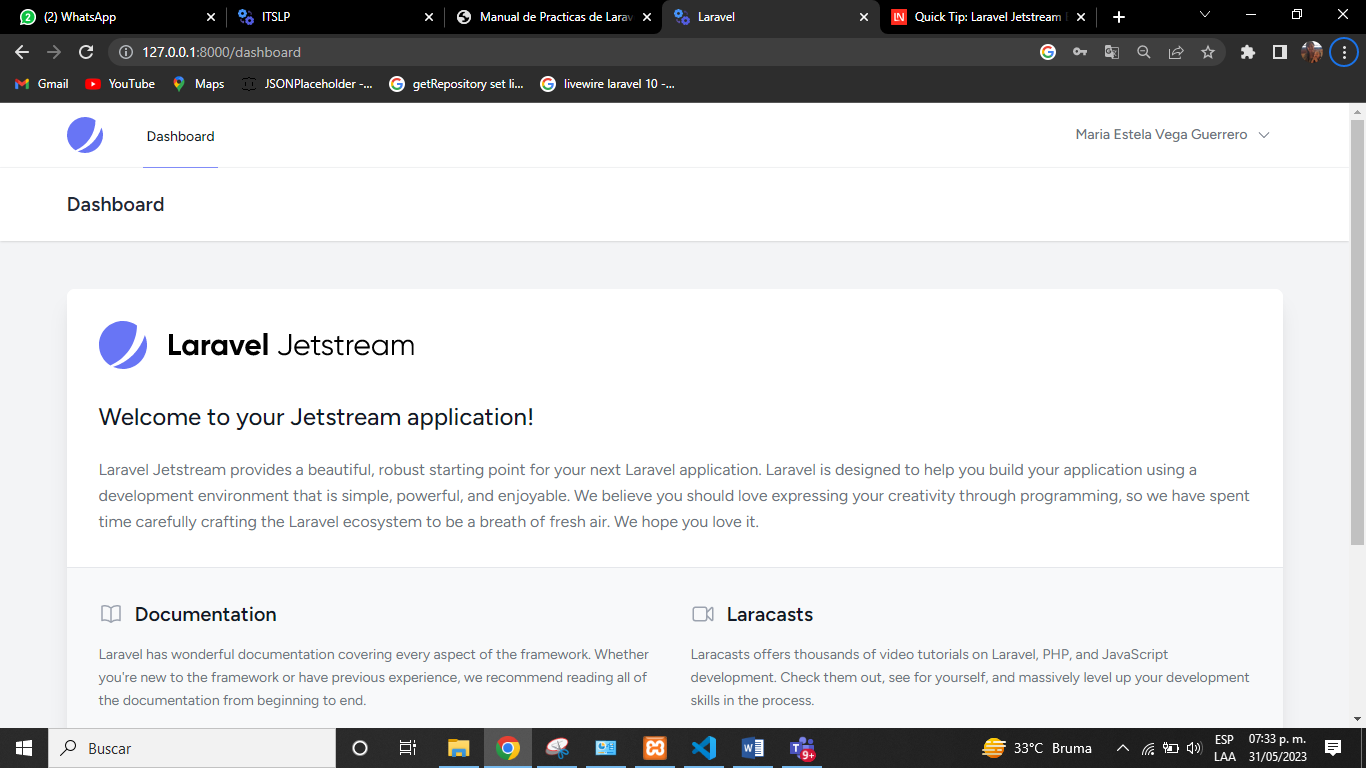


Continuando en body, se agrega una sección que será remplazada con el contenido de quien llame el layaut app



Y finalmente una directiva de regla donde se incluyen los scripts compilados de livewire, línea 43

Así luce la página de dashboard explicada

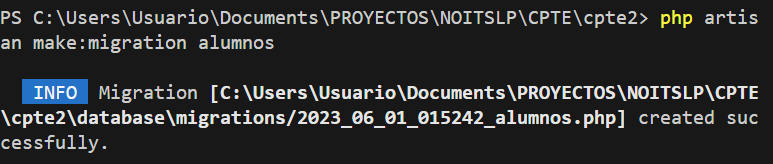


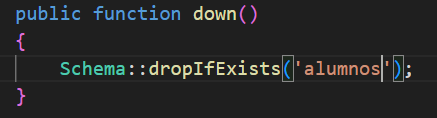
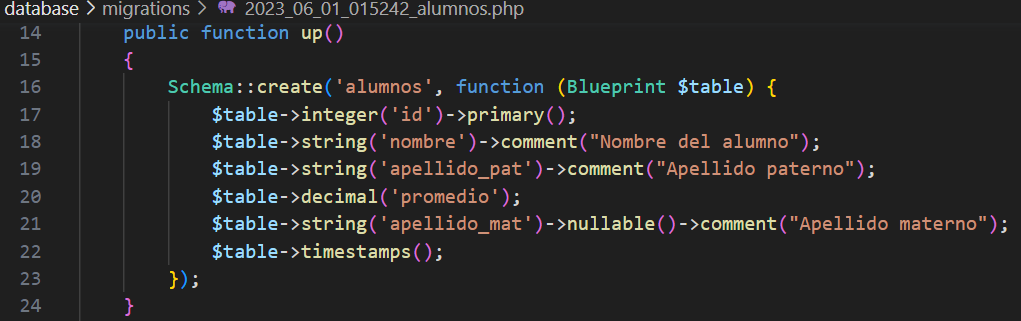
### 3.1.3 Tabla alumnos

Iniciamos por crear la migración de la tabla alumnos y de migrarla a la base de datos

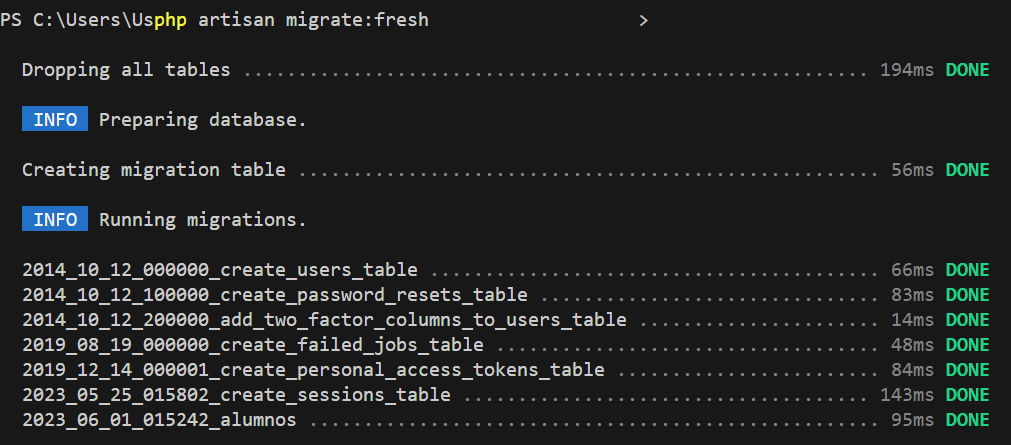
Primero en el archivo .env debe de haber una BD configurada

Podemos ayudarnos del comando php artisan make:migrate alumnos

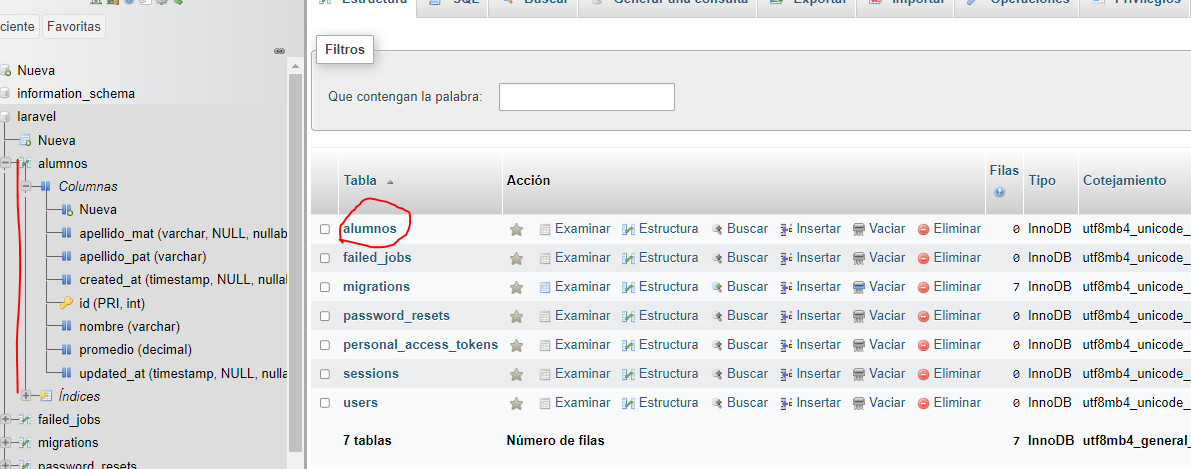




Refrescamos la base de datos, nos podemos ayudar con el comando php artisan migrate:fresh para actualizar las tablas e incluir la nueva tabla.

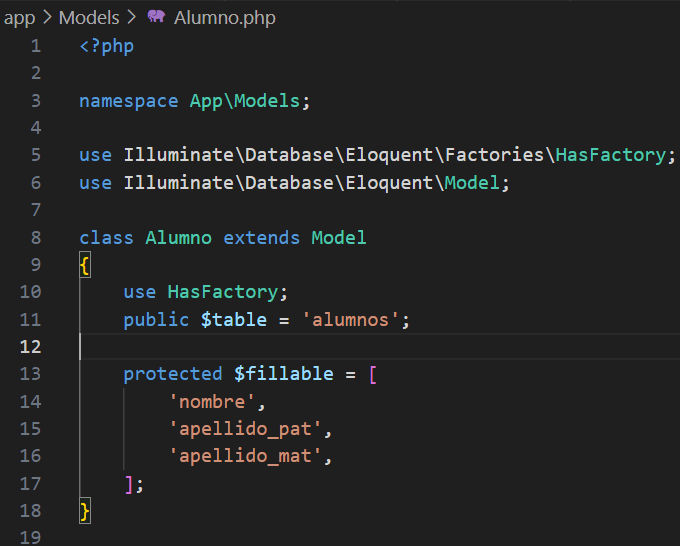


Verificamos en la base de datos:



En Laravel se requiere acceder a las tablas mediante un modelo por lo que para crear el modelo damos la instrucción: *php artisan make:model Alumno*

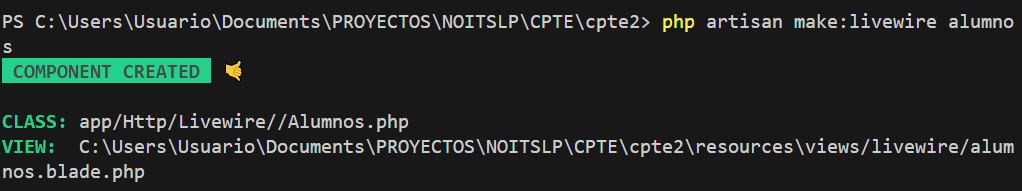
Al editar el archivo nos aseguramos de decirle cuales son los campos que se van a llenar por parte de formularios.



### 3.1.4 Creación del componente livewire alumnos

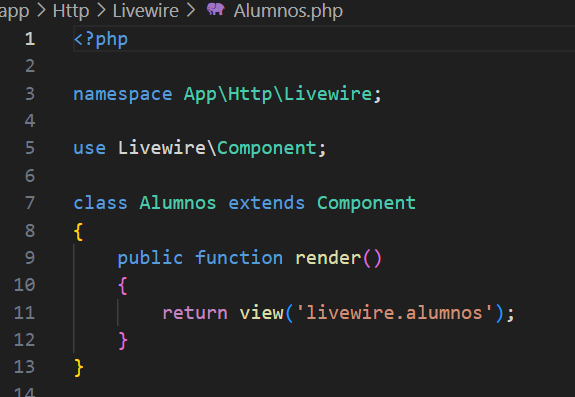
Creamos el compoente alumnos, que se refiere a una clase y una vista que quedan ligados con elementos de javascript, pero programados en php y Blade.

Podemos ayudarnos con el comando: *php artisan make:livewire alumnos*



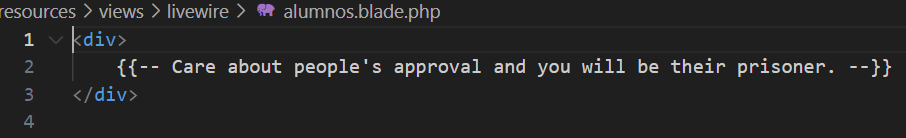
Esto nos crea dos elementos una clase Alumnos y una vista alumnos

Veamos que contiene la clase Alumnos



Como vemos hereda de una clase preparada al instalar livewire Component y un método render que será ejecutado cuando se llame ésta clase, como vemos el método render llama a una vista alumnos.

Vista alumnos



La cual tiene algo que es muy importante la división principal de la vista, dentro se podrá colocar lo que se desee.

Para llamar a la clase y la clase a la vista, se requiere de un evento, puede ser un botón en el menú que llame a una ruta, para ello es necesario agregarla.

### 3.1.5 Ruta para llamar al componente

Agregamos una ruta dando la parte final de la url, la clase y el name de la ruta. Algo así:

Route::get('alumnos', \App\Http\Livewire\Alumnos::class)->name('alumnos');

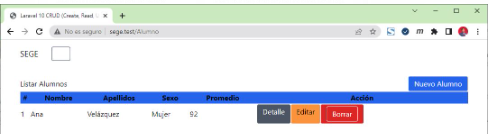
### 3.1.6 Modificando la clase/vista

En la clase modificamos la estructura considerando las siguientes características:

1. Identificar los atributos que serán solicitados en la vista
2. Identificar los métodos o acciones que serán solicitados en la vista

Normalmente el desarrollo de sistemas requiere de un análisis y diseño del sistema, en su sección de diccionario de datos, se definen las tablas y relaciones que se van a requerir, en la sección de diseño se definen entre otros los mockups de las pantallas que realizaremos en el mapa de navegación.

Un **mockups** de alumnos podría ser el siguiente:



Podemos observar las siguientes características:

* Un botón para indicar que deseamos generar un nuevo registro
* Se requiere de una lista de todos los registros ya agregados, donde se coloquen algunos datos de los mismos y al final de cada renglón un botón para permitir editar y otro para permitir borrar
* Las acciones de crear, editar y borrar
* Estilos y texto para formar la interfaz

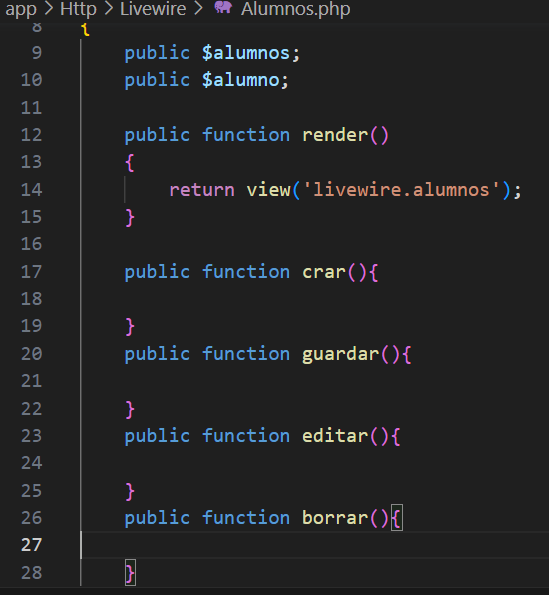
Por lo que en la clase debemos identificar las variables que se desean ver en la vista y estas pueden ser las siguientes

|  |
| --- |
| public $alumnos //Donde guardaremos la colección de registros que será pintada en la vista  public $alumno //Donde guardamos un objeto específico o sea un registro x de alumno, que nos servirá por ejemplo para editarlo. |

También en la clase identificaremos los métodos que pueden ser llamados en la vista y pueden ser los siguientes:

|  |
| --- |
| public crear() //Para indicar que se solicita crear un nuevo registro alumno un nuevo objeto de la clase o modelo Alumno  public guardar() //Para indicar que se guarde la información capturada al crear el registro alumno y se almacenará por medio del model Alumno en la tabla de la BD alumnos  public editar() //Donde responderá al botón editar de un registro en especial, por lo que se requiere que llegue un parámetro que identifique el renglón. El método localizará el registro y lo enviará a la vista por medio de la variable $alumno y llamará la vista por medio de una ventana modal.  public guardar() //El mismo método que sirve para guardar un registro nuevo nos sirve par almacenar un registro editado. Aquí se pueden validar los campos por medio de rules y mensagge  public close() //Para cerrar cualquier ventana modal ya sea la q crea o la que edita.  Public borrar() //Para borrar un registro específico por lo que requiere de un parámetro con el id del registro |

Podemos ir estructurando la clase



Detalle del método render

Vamos a acceder los registros de alumnos *$alumnos=Alumno::all();*

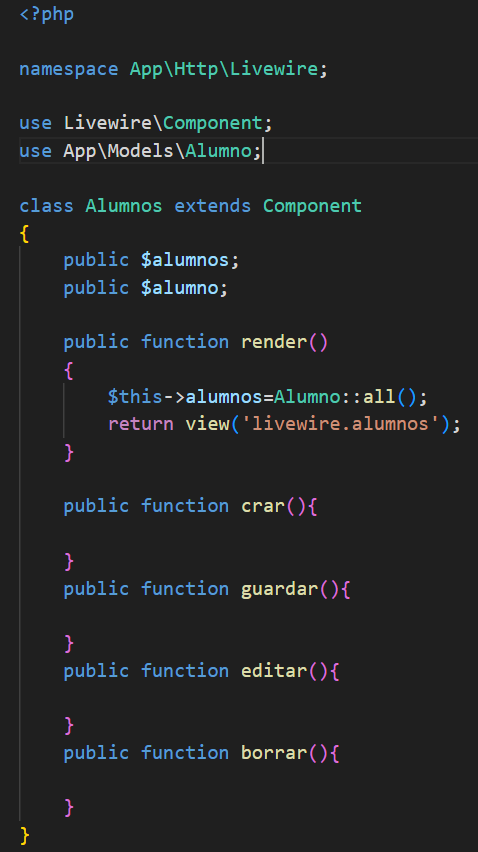
Sin embargo si yo quiero usar éstos datos en la vista y hacer referencia a la variable pública, esto se hace con $this



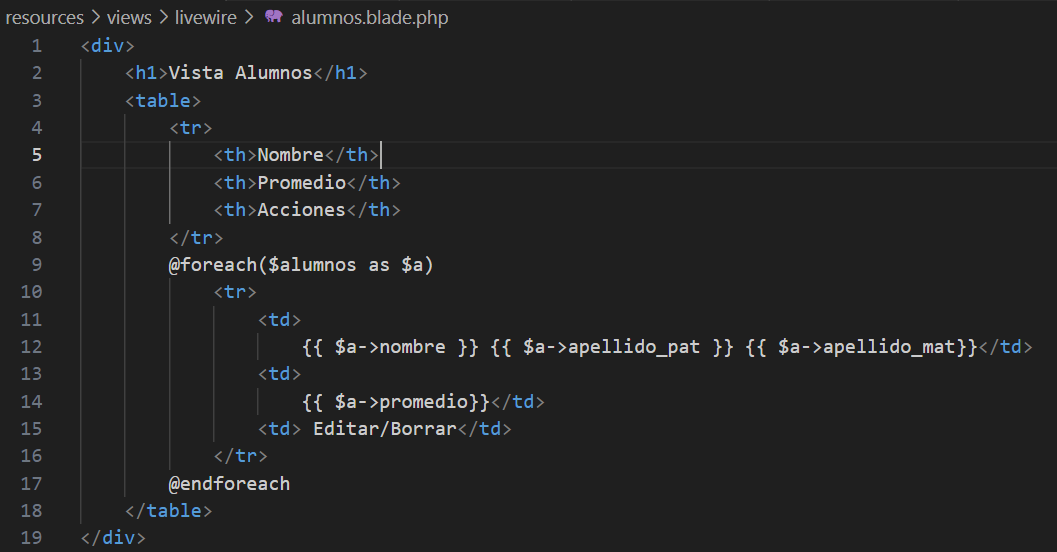
Al utilizar el modelo Alumno, se requiere indicar al inicio el uso de los modelos



Una primera vista de la clase se muestra a continuación:

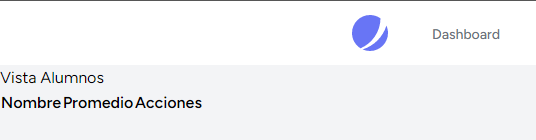


Por otra parte el diseño de la vista la vamos creciendo poco a poco para ir aprendiendo y ver que resultados va teniendo



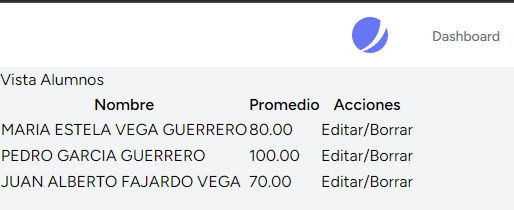
Podemos hacer una revisión en ésta etapa y si tiene registros la tabla alumnos, se pueden ver.

Para poder visualizar el avance, laravel responde a las rutas que se llaman, si recordamos la ruta alumnos es /alumnos, entonces probamos:



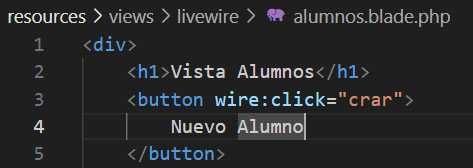
Agregamos directamente algunos registros en la base de datos y volvemos a refrescar.





**Nuevo**

Agregamos en la vista el botón para generar un evento de solicitud para un nuevo alumno

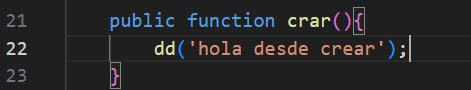


El atributo *wire:click* es una instrucción de livewire para ejecutar el método que se determina en el valor, ya que la clase y la vista están ligados.

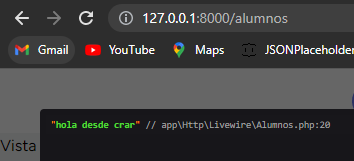
El atributo *wire:submit* igual a un nombre de método, sirve para ejecutar el método cuando se de guardar

El atributo *wire:model* es para relacionar el valor que llega de la clase en la vista y viceversa.

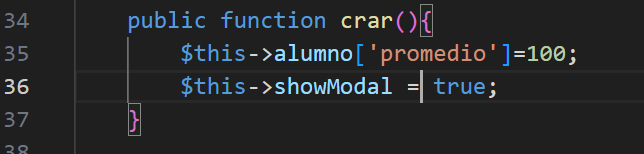
Continuando con la acción del click sobre el botón Nuevo. Podemos ayudarnos con el comando *dd(‘hola desde crear’),* para detener el sistema y ver que si llega al método.



Al ejecutar con click en botón *Nuevo*, se verá así:



Para crear necesitamos que la variable $alumno se encuentre vacía y podemos generar un objeto con algunos campos predeterminados, por ejemplo supongamos que el promedio es un calculo previo, por lo que le pondremos un valor por default.



Asegurándonos que la variable pública $alumno sea un arreglo.

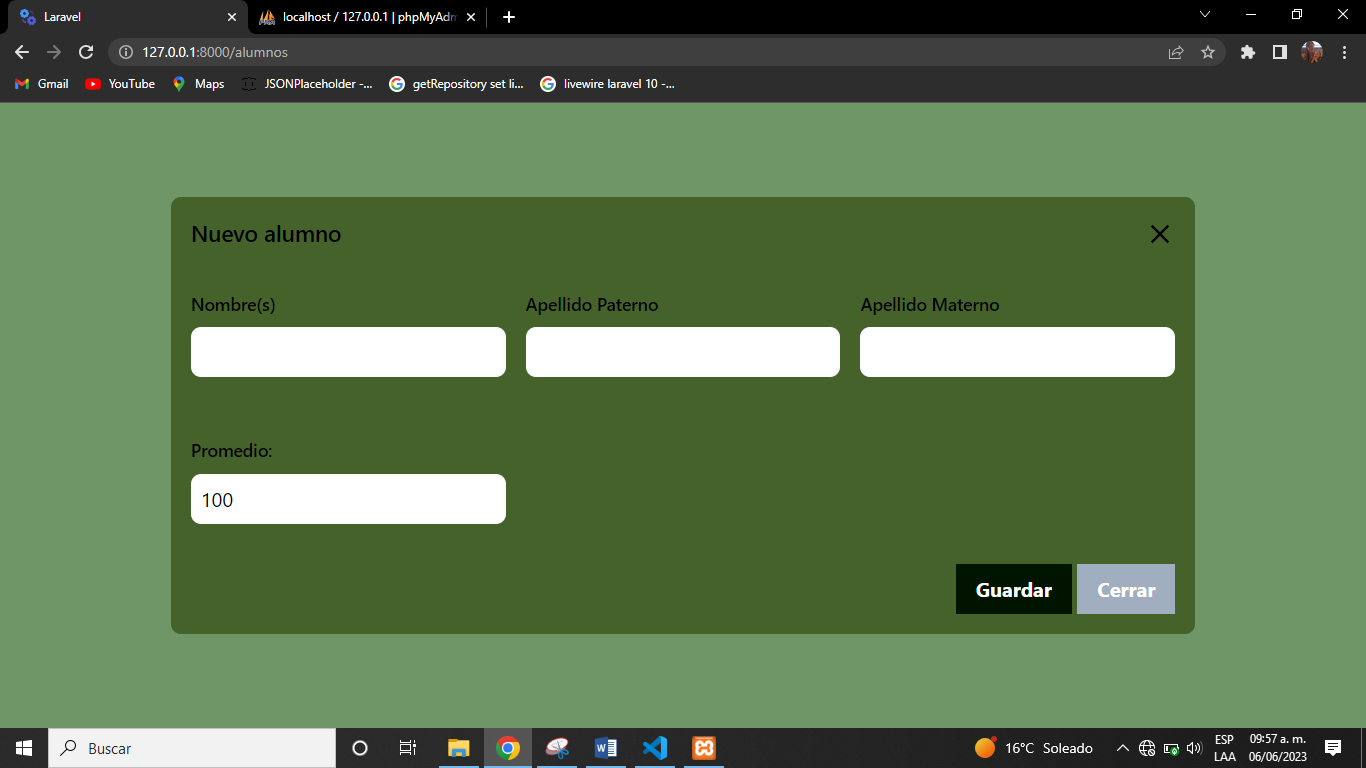


En la línea 35, agregamos al arreglo indexado el valor 100 en promedio, solo para probar que se cuente con un valor previo.

Al asignar el valor *true* a la variable *showModal*, la ventana modal se activa, como muestra la siguiente figura.

Antes de probar y para que funcionen los estilos hacer dos cosas:

* Activar el *npm run dev*
* Tener el archivo de configuración en raíz del proyecto *tailwind.config.js*



El código de la vista en la sección de *Nuevo* o *Editar* es la siguiente:

|  |
| --- |
| <div          class="@if (!$showModal) hidden @endif flex items-center justify-center fixed left-0 bottom-0 w-full h-full bg-rt\_s\_light bg-opacity-90">          <div class="rounded-lg w-3/4 bg-rt\_p\_light">              <form wire:submit.prevent="guardar" class="w-full">                  <div class="flex flex-col items-start p-4">                      <div class="flex items-center w-full border-b pb-4">                          <div class="rt\_s\_text font-medium text-lg">{{ $alumnoId ? 'Editar Alumno' : 'Nuevo alumno' }}</div>                          <svg wire:click="close" class="ml-auto fill-current w-6 h-6 cursor-pointer rt\_s\_text"                              xmlns="http://www.w3.org/2000/svg" viewBox="0 0 18 18">                              <path                                  d="M14.53 4.53l-1.06-1.06L9 7.94 4.53 3.47 3.47 4.53 7.94 9l-4.47 4.47 1.06 1.06L9 10.06l4.47 4.47 1.06-1.06L10.06 9z" />                          </svg>                      </div>                      <div class="grid grid-cols-3 gap-4 w-full">                          <div class="py-4   mb-4">                              <label class="rt\_primary block font-medium text-sm  rt\_s\_text" for="title">                                  Nombre(s)                              </label>                              <input wire:model="alumno.nombre"                              class="mt-2 text-sm sm:text-base pl-2 pr-4 rounded-lg border border-gray-400 w-full py-2 focus:outline-none focus:border-blue-400" />                              @error('alumno.nombre') <span class="text-sm text-red-500 ml-1"> {{$message}} </span>                              @enderror                          </div>                          <div class="py-4   mb-4">                              <label class="rt\_primary block font-medium text-sm  rt\_s\_text" for="title">                                  Apellido Paterno                              </label>                              <input wire:model="alumno.apellido\_pat"                              class="mt-2 text-sm sm:text-base pl-2 pr-4 rounded-lg border border-gray-400 w-full py-2 focus:outline-none focus:border-blue-400" />                              @error('alumno.apellido\_pat') <span class="text-sm text-red-500 ml-1"> {{$message}} </span>                              @enderror                          </div>                          <div class="py-4   mb-4">                              <label class="rt\_primary block font-medium text-sm  rt\_s\_text" for="title">                                  Apellido Materno                              </label>                              <input wire:model="alumno.apellido\_mat"                              class="mt-2 text-sm sm:text-base pl-2 pr-4 rounded-lg border border-gray-400 w-full py-2 focus:outline-none focus:border-blue-400" />                              @error('alumno.apellido\_mat') <span class="text-sm text-red-500 ml-1"> {{$message}} </span>                              @enderror                          </div>                      </div>                      <div class="grid grid-cols-3 gap-4 w-full">                          <div class="py-4   mb-4">                              <label class="rt\_primary block font-medium text-sm  rt\_s\_text" for="title">                                  Promedio:                              </label>                              <input wire:model="alumno.promedio"                              class="mt-2 text-sm sm:text-base pl-2 pr-4 rounded-lg border border-gray-400 w-full py-2 focus:outline-none focus:border-blue-400" />                              @error('alumno.promedio') <span class="text-sm text-red-500 ml-1"> {{$message}} </span>                              @enderror                          </div>                      </div>                      <div class="ml-auto">                          <button class="bg-rt\_p\_dark hover:bg-rt\_p\_dark text-rt\_p\_text font-bold py-2 px-4 rounded"                              type="submit">{{ $alumnoId ? 'Guardar cambios' : 'Guardar' }}                          </button>                          <button class="bg-gray-500 hover:bg-rt\_p\_dark text-rt\_p\_text font-bold py-2 px-4 rounded"                              wire:click="close" type="button" data-dismiss="modal">Cerrar                          </button>                      </div>                  </div>              </form>          </div>      </div> |

Los estilos por el momento no son estudiados en ésta práctica.

Los puntos más importantes de ésta división son los siguientes:

* Se abraza toda la vista modal en una división
* En la clase de la división principal se consulta por medio de la variable $showModal si esta activa o no activa con sus valores respectivos de *true* o *false*. Con ello se oculta en false.
* Todos los elementos del formulario se encuentran abrazados dentro de la definición del formularios por medio de la etiqueta *form*.
* La acción que se ejecutará en el submit del form se define por medio del atributo wire:submit.prevent
* Se asocia el icono X a un método llamado *close*, por medio del atributo de livewire *wire:clck*
* El título de la ventana es variable y puede ser “Nuevo Alumno o Editar Alumno”, de acuerdo al resultado de la expresión {{ $alumnoId ? 'Editar Alumno' : 'Nuevo alumno' }}
* Todos los campos input de captura de texto, son relacionados con el nombre de una variable en el atributo *wire:model*, con ello se relaciona con el control de esas variables en la clase livewire relacionado con la vista.
* Todo campo que se desee validar deberá de llamar a las reglas y mensajes de validacón
* El formulario deberá contener el botón submit.

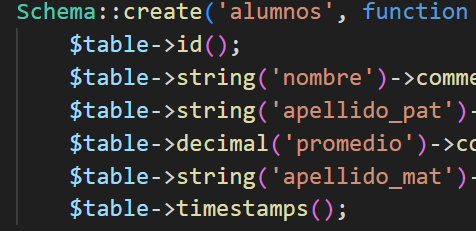
**Validación en el formulario**

Para generar una validación en el formulario deberemos de considerar los siguientes:

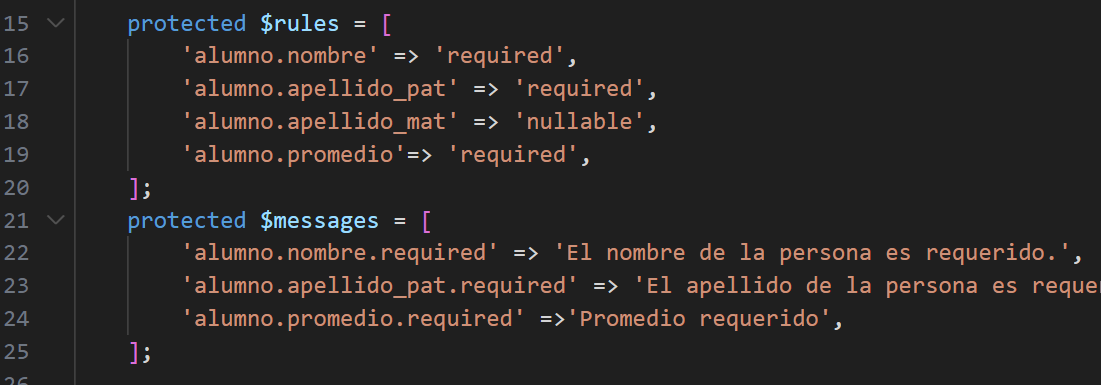
1. Las reglas con las que se creo la tabla
2. Definir las reglas y los mensajes de las reglas en la clase
3. Asociar las reglas en la vista
4. Llamar a la validación antes de guardar los cambios.

Ejemplos:

Reglas en la tabla:



Reglas y mensajes en la clase:

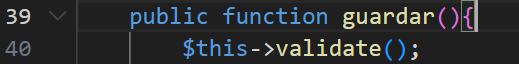


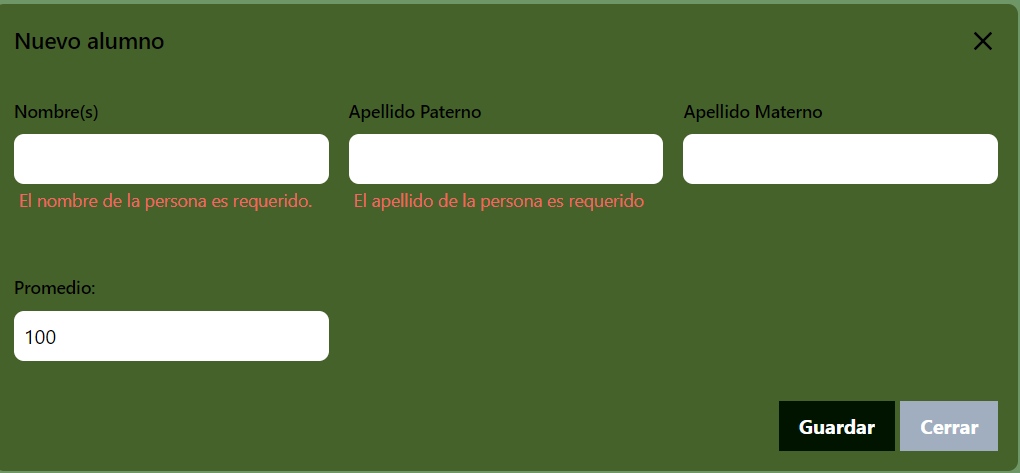
Asociar las reglas en la vista

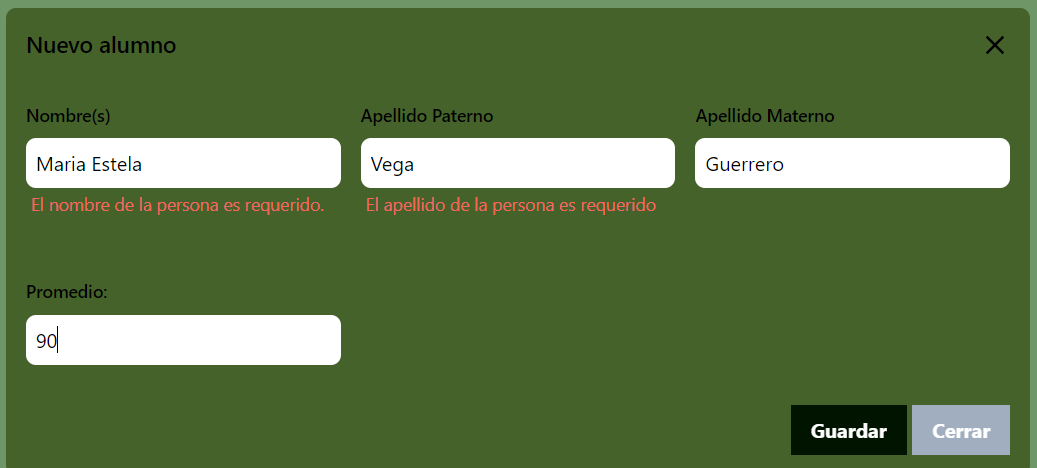
@error('alumno.promedio') <span class="text-sm text-red-500 ml-1"> {{$message}} </span>

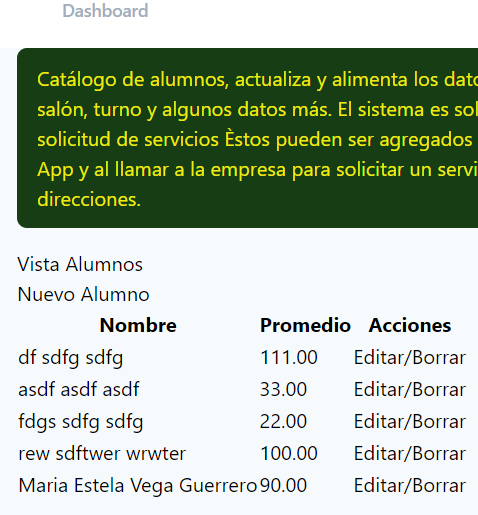
@enderror

Llamar a la validación en nel método antes de guardar



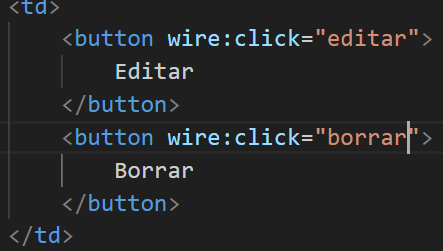






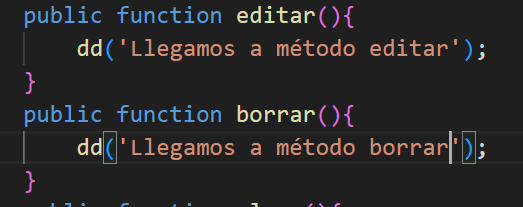
**Editar alumno**

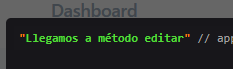
Agregamos dos botones uno para editar y otro para borrar en la vista



Si recordamos el atributo *wire:click* igual a un método, significa que cuando el usuario de click en ese botón se ejecutará el método de la clase *editar* o *borrar*

Podemos usar el dd solo para demostrar que estamos bien ubicados:





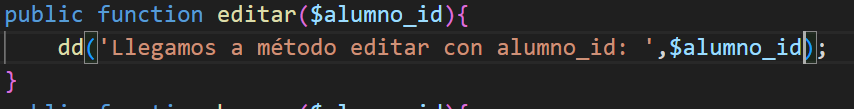


Una vez que comprobamos que si llegará a los métodos correctos, debemos enviar por medio de un parámetro el número id del registro de la tabla alumnos. Para ello agregamos algo así:

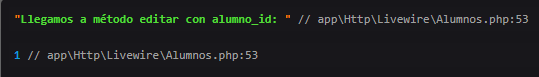




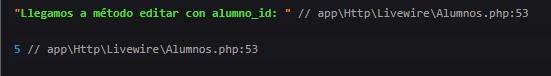
Vamos a revisar si en verdad llegan los ids por medio del comando dd, algo así:



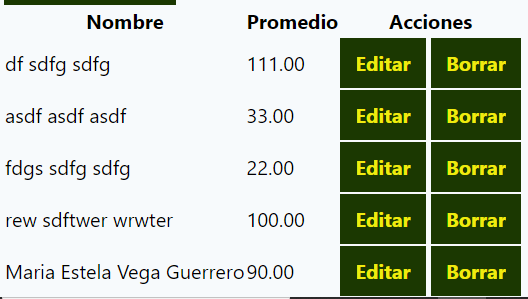
Y lo vemos en ejecución, luego de dar click en el primer registro, así:



Para el último de nuestros registros se ve así:

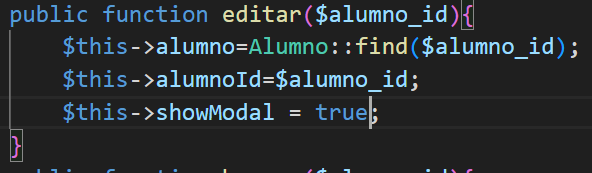


Que corresponden a los registros de la base de datos, así:



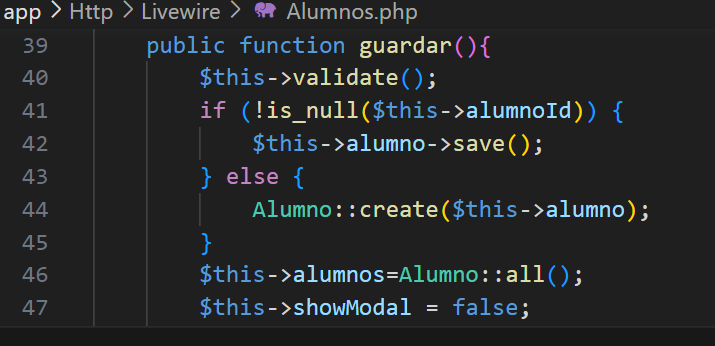
Para editar un registro debemos traerlo de la tabla correspondiente y pasarlo a la ventana modal con los valores correspondientes, esperar los cambios, hacer las validaciones y realizar la actualización del cambio.

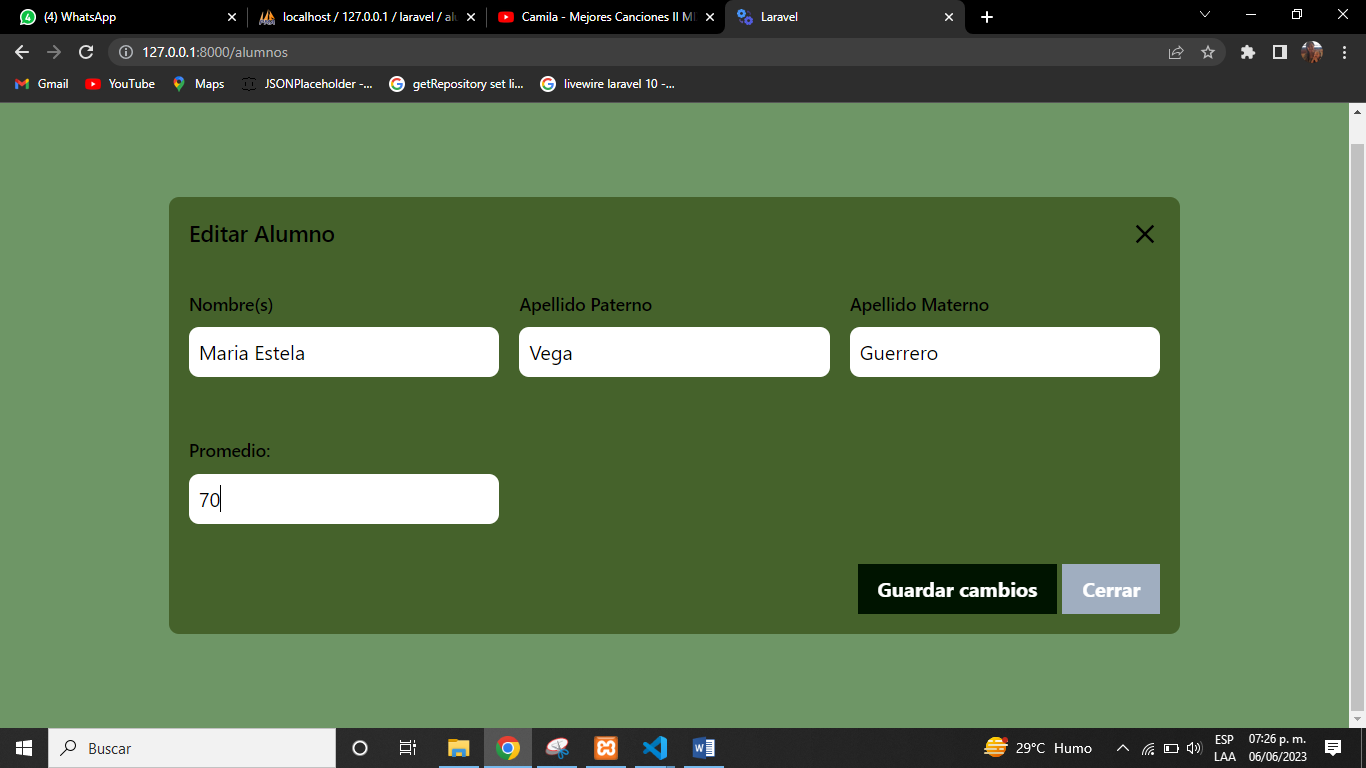
1.\_ Localizamos el registro, agregamos el id a la variable pública $alumnoId para identificar el registro y activamos la modal.

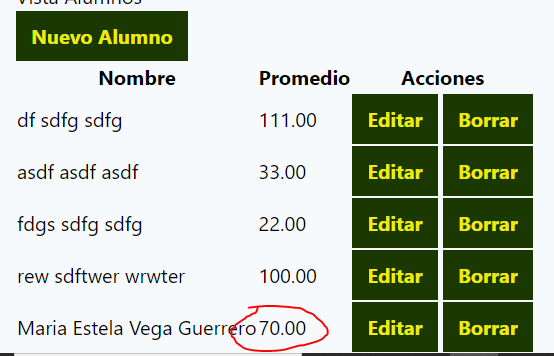




2.\_ Actualizamos los datos cambiados en el método guardar, que ya antes al crear lo dejamos preparado para guardar la edición.







**Borrar alumno**

