

NOTIFICACIONES EN TIEMPO REAL.

Ahora usaremos un nuevo concepto que son Broadcasting (transmisión) y las notificaciones, estas las utilizaremos para monitorear en tiempo real las ventas y compras realizadas en nuestra aplicación.

Enlace al Broadcasting: <https://laravel.com/docs/5.6/broadcasting>

Enlace a las notificaciones: <https://laravel.com/docs/5.6/notifications>

Sin embargo Laravel también nos facilita la transmisión de nuestros eventos a través de una conexión con WebSockets, es decir comunica datos del lado del servidor a nuestra aplicación utilizando JavaScript (lado del cliente).

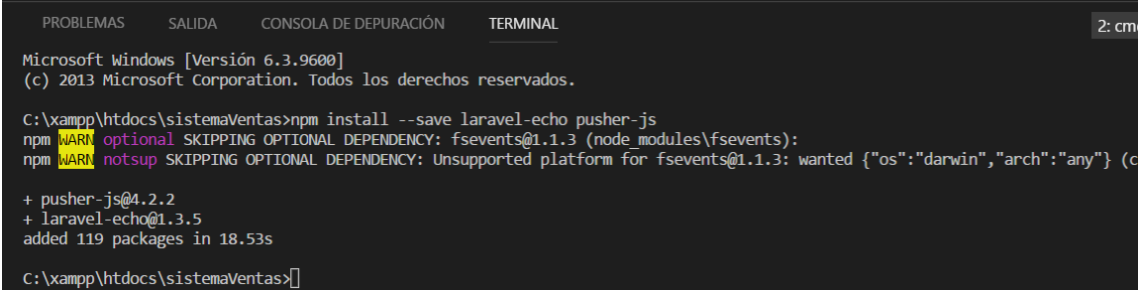
Laravel admite diferentes controladores de transmisión de eventos listos para usar como son: **Pusher, Redis y un controlador de desarrollo local.**

Para suscribirnos a los canales (como son Pusher y Redis) y escuchar eventos transmitidos por nuestra aplicación Laravel de una manera más sencilla, utilizaremos **“Laravel Echo”** que es una biblioteca de JavaScript que facilita la suscripción a canales y escuchar eventos transmitidos por laravel, para instalar Laravel Echo utilizaremos NPM, en este ejemplo también utilizaremos el paquete “Pusher-js” ya que transmitiremos los eventos utilizando “Pusher”.

Enlace: <https://laravel.com/docs/5.6/broadcasting#installing-laravel-echo>

Instalar Laravel Echo Simple: `npm install --save laravel-echo`

Instalar Laravel Echo + Paquete Pusher: `npm install --save laravel-echo pusher-js`



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  2: cm
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

C:\xampp\htdocs\sistemaVentas>npm install --save laravel-echo pusher-js
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.3 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.3: wanted {"os":"darwin","arch":"any"} (c
+ pusher-js@4.2.2
+ laravel-echo@1.3.5
added 119 packages in 18.53s

C:\xampp\htdocs\sistemaVentas>
```

Una vez que se haya instalado Laravel Echo, está listo para crear una nueva instancia de Echo en el JavaScript de su aplicación. Un gran lugar para hacer esto es en la parte inferior del archivo que se incluye con el marco de Laravel (es decir el archivo “Bootstrap”, donde importamos AXIOS, y todo lo relacionado a JavaScript), lo que debemos hacer es descomentar las líneas que vienen comentadas por defecto en la parte inferior:

Ubicación: “resources/assets/js/bootstrap.js”

```
import Echo from 'laravel-echo'

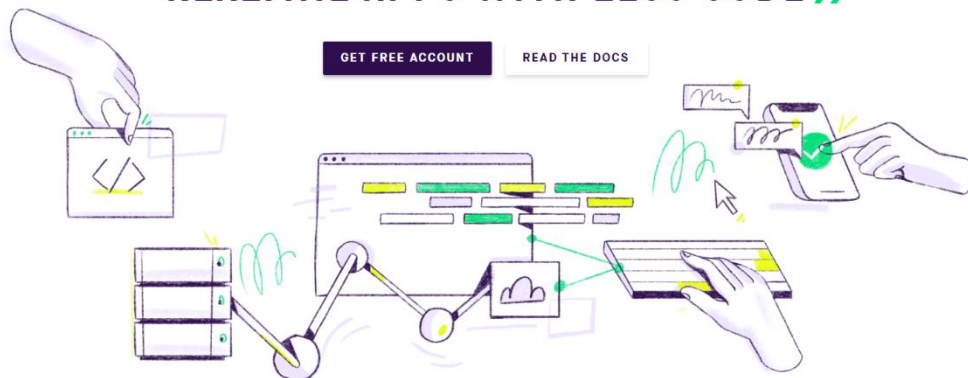
window.Pusher = require('pusher-js');

window.Echo = new Echo({
  broadcaster: 'pusher',
  key: 'your-api-key',
  cluster: 'us2',
  encrypted: true
});
```

Como pueden observar la instancia de Laravel Echo nos solicita credenciales como la KEY y Clúster de Pusher, para ello necesitamos ingresar al sitio web oficial de Pusher: <https://pusher.com/channels>, y nos creamos una cuenta.



HOSTED APIs TO BUILD REALTIME APPS WITH LESS CODE //



Create a FREE account

...and build scalable realtime apps in seconds

Sign up with ...

GitHub

Google

Or sign up with email

Email address

Password

Company name

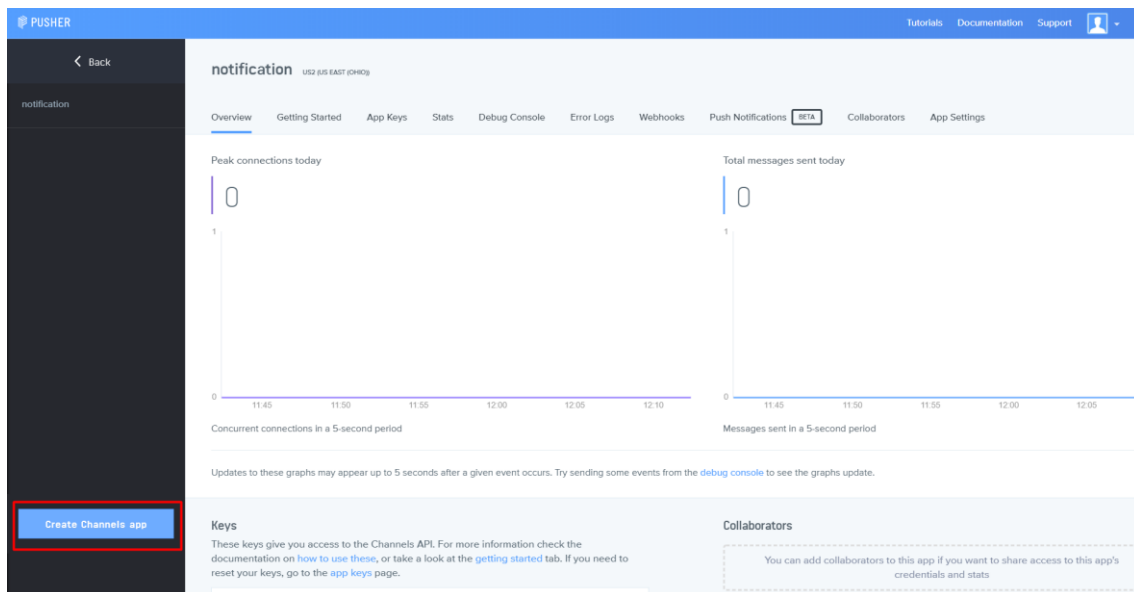
Create a FREE account

You agree to the Pusher [Terms of Service](#) and [Privacy Policy](#).

Una vez ingresado en el sistema, nos aparece la siguiente interfaz, y lo siguiente a realizar es presionar el botón de crear una nueva Aplicación de Canal (Channels Apps).



Presionamos en crear nuevo canal de aplicación.



Y nos aparecerá un formulario, el cual está conformado por 5 pasos:

1. Asignar un nombre a la aplicación
2. Seleccionar un clúster (lo deje por defecto)
3. Seleccionar la tecnología del lado del cliente.
4. Seleccionar la tecnología del lado del servidor.
5. Explicar brevemente de que tratara tu canal de aplicación (opcional).

Y por último presionamos en “Create my App”

Create your Channels app

Create a Channels app to generate your unique credentials. You can create as many apps as you need.

1 Name your app ?

sistema-ventas

2 Select a cluster ?

ap2 (Asia Pacific (Mumbai))

☐ Create apps for multiple environments?

Customize your getting started guide (optional).

3 What's your front-end tech?

Android

Angular

Apple

Flutter

JS

Next.js

React Native

4 What's your back-end tech?

Node.js

Django

Python

Rails

PHP

Spring

Vue.js

5 Tell us about your app (optional)

What do you intend to build with this Channels app?

Estoy construyendo una aplicación de compra y ventas

Create my app

Una vez creada la aplicación nos enviara al panel de administración que nos brindara todos los datos necesarios para incorporarlo en nuestra aplicación de Laravel.

PUSHER

Tutorials Documentation Support

sistemaVentas

US2 (US EAST (OHIO))

Overview

Getting Started

App Keys

Stats

Debug Console

Error Logs

Webhooks

Push Notifications

Collaborators

App Settings

1 Add this to your client

JavaScript

iOS (Obj-C)

iOS (Swift)

Android (Java)

React Native

2 Add this to your server

Curl

Ruby

Rails

Node.js

ASP.NET MVC

Python

PHP

Go

Java

CLI

.net

This page includes the JavaScript below. Connection state: **Connected**

```

<DOCTYPE html>
<html>
<head>
<title>Pusher Test</title>
<script src="https://js.pusher.com/4.1/pusher.min.js"></script>
</script>

// Enable pusher logging - don't include this in production
Pusher.logToConsole = true;

var pusher = new Pusher('d8bed81c35fbaf5e2a4b', {
  cluster: 'us2',
  encrypted: true
});

var channel = pusher.subscribe('my-channel');
channel.bind('my-event', function(data) {
  // ...

```

Install via Composer:

```
$ composer require pusher/pusher-php-server
```

Import the Pusher package and trigger your first event:

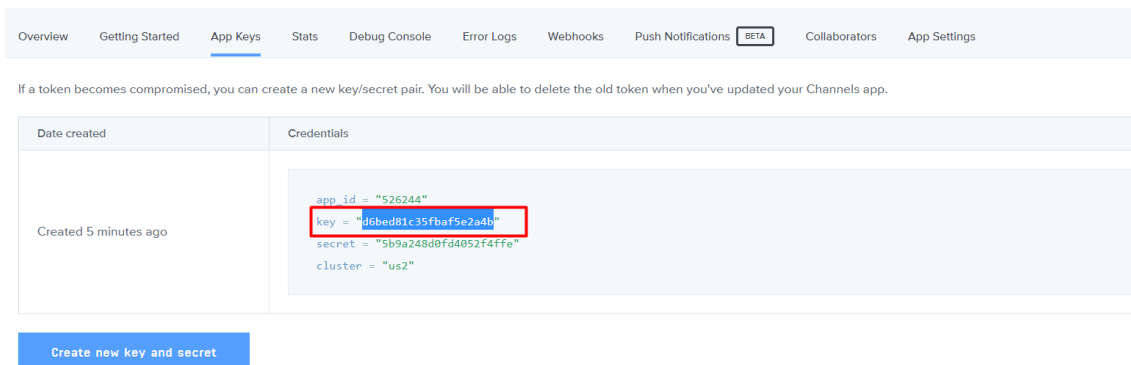
```

<?php
require __DIR__ . '/vendor/autoload.php';

$options = array(
    'cluster' => 'us2',
    'encrypted' => true
);
$pusher = new Pusher\Pusher(
    'd8bed81c35fbaf5e2a4b',
    '569a248d0f4d852f4ffe',
    '526244',
    $options
);

```

Vamos a la sección de “App Keys” y copiamos la parte que dice “key” y “clúster”



Overview Getting Started **App Keys** Stats Debug Console Error Logs Webhooks Push Notifications **BETA** Collaborators App Settings

If a token becomes compromised, you can create a new key/secret pair. You will be able to delete the old token when you've updated your Channels app.

| Date created | Credentials |
|-----------------------|---|
| Created 5 minutes ago | <pre>app_id = "526244" key = "d6bed81c35fbaf5e2a4b" secret = "5b9a248d0fd4052f4ffe" cluster = "us2"</pre> |

Create new key and secret

Y la pegamos en nuestro archivo “Bootstrap” de nuestra aplicación.

Ubicación: “resources/assets/js/bootstrap.js”

```
import Echo from 'laravel-echo'

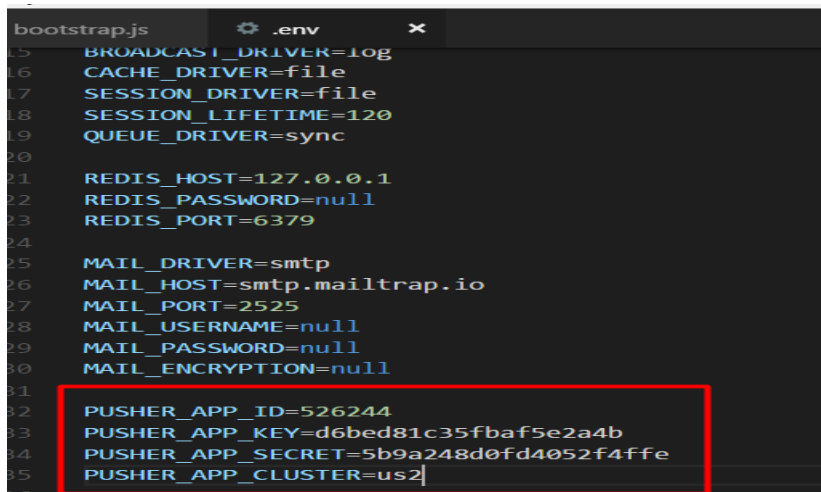
window.Pusher = require('pusher-js');

window.Echo = new Echo({
  broadcaster: 'pusher',
  key: 'd6bed81c35fbaf5e2a4b',
  cluster: 'us2',
  encrypted: true
});
```

Además una vez instalado Laravel Echo nos dice que necesita acceder al token CSRF de la sesión actual. Debe verificar que el elemento head de HTML de nuestra aplicación defina una meta etiqueta que contenga el token CSRF.

```
<meta name="csrf-token" content="{ {{ csrf_token() }}">
```

Así como también debemos pegar estas credenciales en nuestro archivo “.env” que se encuentra ubicado en la raíz de nuestro proyecto de Laravel, dentro de él en la parte inferior se ubica las variables de entorno de Pusher.



```
bootstrap.js .env x
15 BROADCAST_DRIVER=log
16 CACHE_DRIVER=file
17 SESSION_DRIVER=file
18 SESSION_LIFETIME=120
19 QUEUE_DRIVER=sync
20
21 REDIS_HOST=127.0.0.1
22 REDIS_PASSWORD=null
23 REDIS_PORT=6379
24
25 MAIL_DRIVER=smtp
26 MAIL_HOST=smtp.mailtrap.io
27 MAIL_PORT=2525
28 MAIL_USERNAME=null
29 MAIL_PASSWORD=null
30 MAIL_ENCRYPTION=null
31
32 PUSHER_APP_ID=526244
33 PUSHER_APP_KEY=d6bed81c35fbaf5e2a4b
34 PUSHER_APP_SECRET=5b9a248d0fd4052f4ffe
35 PUSHER_APP_CLUSTER=us2
36
```

Como pueden ver hemos agregado las variables de entorno de Pusher en nuestra aplicación Laravel, para ser exactos en el archivo .env, lo siguiente que debemos realizar es instalar el SDK PHP de Pusher en nuestra aplicación ya que este será el que transmita eventos, para ello utilizaremos el administrador de paquetes Composer.

Podemos encontrar el comando para instalar el paquete de 02 maneras.

1. En la documentación oficial de Laravel.
Enlace: <https://laravel.com/docs/5.6/broadcasting#introduction>, en la sección de – Driver Pre Requisitos.

Driver Prerequisites

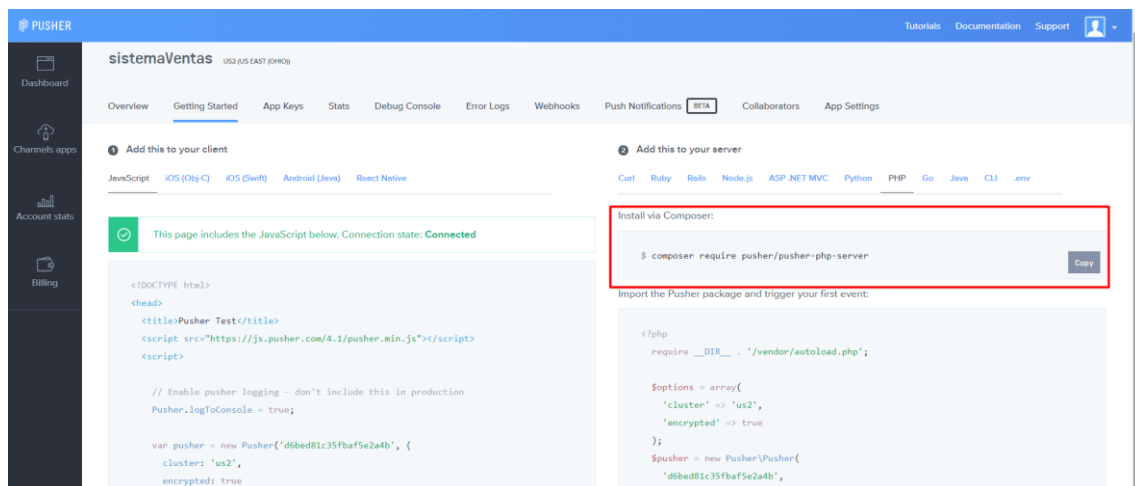
Pusher

If you are broadcasting your events over [Pusher](#), you should install the Pusher PHP SDK using the Composer package manager:

```
composer require pusher/pusher-php-server "~3.0"
```

2. Desde el panel de administración de Pusher
Enlace: https://dashboard.pusher.com/apps/526389/getting_started

```
$ Composer require pusher/pusher-php-server
```



The screenshot shows the Pusher dashboard interface. On the left, there's a sidebar with navigation links like Dashboard, Channels apps, Account stats, and Billing. The main content area is titled 'sistemaVentas' and shows the 'Getting Started' tab. It includes a section for 'Add this to your client' with JavaScript code and a section for 'Add this to your server' with PHP code. A red box highlights the 'Install via Composer' section, which displays the command: `$ composer require pusher/pusher-php-server`. Below this, there's a section for 'Import the Pusher package and trigger your first event:' which shows a PHP code snippet for initializing the Pusher client.

```

C:\xampp\htdocs\sistemaVentas>npm install --save laravel-echo pusher-js
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.3 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.3: wanted {"os":"darwin","arch":"any"}
+ pusher-js@4.2.2
+ laravel-echo@1.3.5
added 119 packages in 18.53s

C:\xampp\htdocs\sistemaVentas>composer require pusher/pusher-php-server
Using version ^3.0 for pusher/pusher-php-server
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing pusher/pusher-php-server (v3.0.3): Downloading (100%)
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover
Discovered Package: fideloper/proxy
Discovered Package: laravel/tinker
Discovered Package: barryvdh/laravel-dompdf
Package manifest generated successfully.

```

Una vez instalado la biblioteca Laravel Echo, el SDK Pusher y configurado nuestros archivos Bootstrap.js y .env, queda configurar las credenciales de Pusher en el archivo de configuración ubicado en el directorio “config/Broadcasting.php” / PUSHER => OPTIONS

Una vez configurado tanto el Laravel Echo como el Pusher, empecemos a programar las notificaciones:

Enlace: <https://laravel.com/docs/5.6/notifications>

Como se explicó anteriormente Laravel proporciona soporte para el envío de notificaciones, en este almacenaremos las notificaciones en una base de datos para que se muestren en la web.

Como explican en la documentación enlace:

<https://laravel.com/docs/5.6/notifications#introduction>

“Las notificaciones deben ser breves, mensajes informativos que avisan a los usuarios de algo que ocurrió en su aplicación. Por ejemplo, si está escribiendo una aplicación de facturación, puede enviar una notificación de “factura pagada” a sus usuarios a través de los canales de correo electrónico, SMS o en la misma aplicación.

Como en este caso las notificaciones se almacenaran en la base de datos necesitaremos una tabla de base datos, esta tabla contendrá la información como el tipo de notificación y los datos JSON personalizados que describen la notificación.

Enlace: <https://laravel.com/docs/5.6/notifications#database-notifications>

Entonces necesitaremos crear una tabla de base de datos para guardar sus notificaciones. Puede utilizar el comando artisan “notifications:table” para generar una migración con el esquema de tabla adecuado.

Comando para crear la migración: `php artisan notifications:table`

Commando para migrar de Nuevo las migraciones: `php artisan migrate`

Si revisamos la migración podrá observar que tiene un campo id, un campo tipo, el tipo de la notificación y la data que contendrá el contenido de la notificación, adicionalmente un campo “read_at” que verifica que la notificación fue leída o no.

```

class CreateNotificationsTable extends Migration
{
    /** ...
    public function up()
    {
        Schema::create('notifications', function(Blueprint $table) {
            $table->uuid('id')->primary();
            $table->string('type');
            $table->morphs('notifiable');
            $table->text('data');
            $table->timestamp('read_at')->nullable();
            $table->timestamps();
        });
    }
}

```

Como utilizaremos una migración que posteriormente se convertirá en una tabla de la base de datos, necesitaremos consultar dichos datos en la base de datos es por ello que también crearemos el modelo “Notification” mediante el comando “make:model”

Comando para crear el comando: `php artisan make:model Notification`

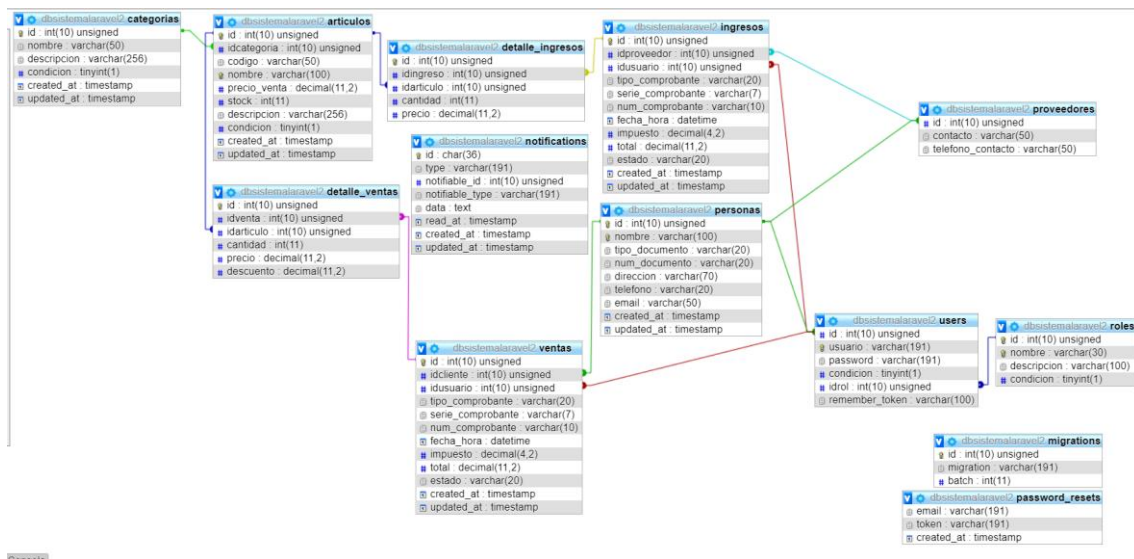
Comando para migrar las tablas: `php artisan migrate:fresh`

```

C:\xampp\htdocs\sistemaVentas>php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table
Migrating: 2018_02_01_171225_create_categorias_table
Migrated: 2018_02_01_171225_create_categorias_table
Migrating: 2018_02_22_235257_create_articulos_table
Migrated: 2018_02_22_235257_create_articulos_table

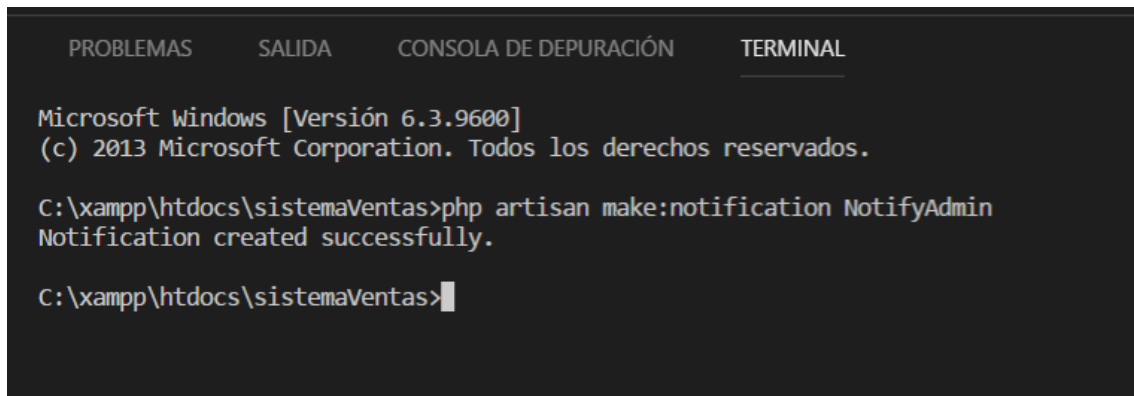
```

Revisemos ahora la estructura de la base de datos.



Ahora procederemos a crear la notificación, en Laravel, cada notificación está representada por una única clase (normalmente almacenada en el directorio “app/Notifications”). No hay que preocuparse si no se encuentra el directorio, se creará al ejecutar el comando de “Artisan make: Notification”:

Comando para crear la notificación: `php artisan make:notification NombreNotificación`



```
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

C:\xampp\htdocs\sistemaVentas>php artisan make:notification NotifyAdmin
Notification created successfully.

C:\xampp\htdocs\sistemaVentas>
```

Este comando colocará una nueva clase de notificación en el directorio “app/Notifications”.



```
9
10 class NotifyAdmin extends Notification
11 {
12     ... use Queueable;
13
14     ... /** ...
19     public function __construct()
20     ... {
21     ... //
22     ... }
23
24     ... /** ...
30     public function via($notifiable)
31     ... {
32     ... return ['mail'];
33     ... }
34
35     ... /** ...
41     public function toMail($notifiable)
42     ... {
```

Cada clase de notificación contiene un método “vía” que determina en que canales se entregara la notificación. Las notificaciones se pueden enviar a los canales “mail”, “database”, “broadcast”, “nexmo” y “slack” (Es decir desde tu aplicación se enviaran desde tu aplicación hacia dichos canales: Eje. Puedo enviar correos electrónicos desde mi app ó puedo enviar mensajes a Slack desde mi app). El método “vía” recibe una instancia “\$notifiable”, que será una instancia de la clase a la que se envía la notificación. Puede utilizar “\$notifiable” para determinar en qué canales se debe entregar la notificación.

Para el presente proyecto no utilizaremos el envío de notificaciones al canal del mail sino lo enviaremos a través de la base de datos entonces modificamos el canal de entrega de la vía que retorna en este caso “mail”. De igual manera no utilizaremos el método “toMail” por la misma

razón que no enviaremos nada a correos electrónicos, entonces lo modificaremos por “toDatabase” (ya que la información será guardada en la DB) y borramos el contenido del método.

```
/** ...
public function via($notifiable)
{
    return ['database'];
}

/** ...
public function toDatabase($notifiable)
{
    return [
    ];
}

/** ...
public function toArray($notifiable)
{
    return [
        //
    ];
}
```

Nota: Además existen un número variable de métodos de creación de mensajes (como toMail, toDatabase, nexmo, slack) que convierten la notificación en un mensaje optimizado para ese canal en particular.

Lo siguiente a realizar es crear una instancia en el constructor y pasarle como parámetros los datos que recibiremos desde nuestro controlador. ¿Qué es lo que necesitamos?

1. El parámetro que necesitaremos será un arreglo de datos, al cual le llamaremos “\$datos”.

```
2. public function __construct(Array $datos)
3.     {
4.         //
5.     }
```

2. Luego necesitamos declarar una variable global para almacenar la instancia del constructor.

```
class NotifyAdmin extends Notification
{
    use Queueable;

    public $GlobalDatos;

    /** ...
    public function __construct(Array $datos)
    {
        //
    }
}
```

3. Luego necesitamos almacenar el arreglo de la instancia del constructor en la variable global, realizamos esto para poder utilizarla (variable global) al enviar información al o los canales seleccionados (hasta el momento database).

```
4. public function __construct(Array $datos)
5. {
6.     $this->GlobalDatos = $datos;
7. }
```

Luego de tener almacenado el arreglo en nuestra variable global, pasamos a enviarle dichos datos a nuestro canal de transmisión de datos (toDatabase).

```
public function toDatabase($notifiable)
{
    return [
        'datos' => $this->GlobalDatos
    ];
}
```

Una vez que tenemos todo listo para enviar una notificación programaremos dicha notificación dentro del controlador de ventas en la acción “store” es decir al momento de realizar una venta

Para ello utilizaremos el **trait Notifiable**, (si bien existe otra manera de enviar notificaciones mediante el uso de la facade o fachada “**Notification**”), debido que este trait viene incluido en el modelo “App\User” por defecto y contiene un método que puede usarse para enviar notificaciones: “**notify**”. *Enlace:* <https://laravel.com/docs/5.6/notifications#sending-notifications>

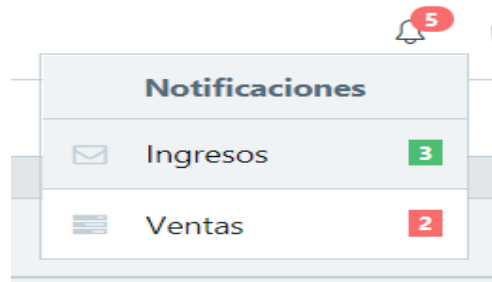
```
<?php
namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    /** ...
    protected $fillable = [
        'id', 'usuario', 'password', 'condicion', 'idrol'
    ];
    ...
```

Lo que nosotros necesitamos visualizar es lo que se visualiza en la plantilla (No olvidar tener levantado el servidor de desarrollo local: “php artisan serve”).



Entonces si analizamos la figura, se necesita el número de ventas e ingresos realizados en el día, adicionalmente un mensaje que diga que uno es un ingreso y otro es una venta, una vez explicado ello nos vamos a programar nuestras consultas. ¿Pero dónde lo haremos?, nosotros queremos que la notificación se actualice cada vez que se realice una compra o venta. Entonces lo haremos como se explicó anteriormente dentro del controlador “VentaController” y “IngresoController” en la acción “store” ya que es la acción para realizar una venta y compra.

Primero vamos al controlador VentaController.

Justo antes del “DB::commit()” que indica que se realizó todo con éxito, realizaremos las consultas.

```
... $numVentas = DB::table('ventas')->count();  
... $numIngresos = DB::table('ingresos')->count();  
... DB::commit();  
... //Recibir el id de la venta para captura y generar  
... return [  
... | 'id' => $venta->id  
... ];
```

Como necesitamos mostrar las ventas y compras del día nos falta un parámetro ahí, y es que necesitamos capturar la fecha actual y enviarlo mediante un “whereDate” en cada consulta. Para ello capturaremos la fecha actual en una variable “\$fechaActual”.

```
$fechaActual= date('Y-m-d');  
$numVentas = DB::table('ventas')->whereDate('created_at', $fechaActual)-  
>count();  
$numIngresos = DB::table('ingresos')->whereDate('created_at',  
$fechaActual)->count();
```

Y estos datos retornan el número de ventas e ingresos del día, así que le daremos forma dentro de un arreglo (recuerde que lo que llega a la notificación es un arreglo).

```
$numVentas = DB::table('ventas')->where('created_at', $mytime)->count();  
$numIngresos = DB::table('ingresos')->where('created_at', $mytime)-  
>count();
```

```
$arregloDatos = [
    'ventas' => [
        'numero' => $numVentas,
        'msj' => 'Ventas'
    ],
    'ingresos' => [
        'numero' => $numIngresos,
        'msj' => 'Ingresos'
    ]
];
```

Ahora una vez que tenemos todas las consultas preparadas en el arreglo, lo que queremos es notificar o avisar a todos nuestros usuarios del sistema sobre la nueva venta o ingreso realizado (es decir al administrador, vendedor y almacenero). Para ello haremos la consulta después de capturar el arreglo.

```
$allUsers = User::all();
```

Importamos la clase User.

```
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Carbon\Carbon;
use App\Venta;
use App\DetalleVenta;
use App\User;
```

Luego de capturar todos los usuarios, debemos recorrer todos los usuarios que vamos a notificar mediante un “foreach”

```
$allUsers = User::all();

foreach ($allUsers as $notificar) {
    # code...
}
```

Dentro del “foreach” debemos ingresar la notificación.

```
foreach ($allUsers as $notificar) {
    User::findOrFail($notificar->id)->notify();
}
```

Como podemos ver tenemos una consulta “User:find” que recibe como parámetro el id de los usuarios a los que se les enviara la notificación y la manera de notificarlos es mediante el método “notify()”.

Nos falta algo más y es que debemos saber que el método “notify” del trait “Notifiable” espera recibir una instancia de una notificación... y cuál es esa notificación, es la notificación que habíamos programado antes es decir “NotifyAdmin”, entonces pasémosle la instancia.

```
foreach ($allUsers as $notificar) {  
    User::findOrFail($notificar->id)->notify(new NotifyAdmin());  
}
```

Así como esta aun no nos funciona ya que nos falta importar dicha clase en nuestro controlador, así que importémosla.

```
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
use Illuminate\Support\Facades\DB;  
use Carbon\Carbon;  
use App\Venta;  
use App\DetalleVenta;  
use App\Notifications\NotifyAdmin;
```

Si no sabes la ruta de la notificación, podemos ir a buscarla dentro del directorio “app\Notifications” y copiar su “namespace”.

```
namespace App\Notifications;  
  
use Illuminate\Bus\Queueable;  
use Illuminate\Notifications\Notification;  
use Illuminate\Contracts\Queue\ShouldQueue;  
use Illuminate\Notifications\Messages\MailMessage;
```

Lo copiamos y lo que nos queda poner es el nombre de la notificación y listo.

```
App\Notifications  
App\Notifications\NotifyAdmin;
```

Ahora antes de enviar la notificación nos falta algo, así que revisemos nuestra clase instanciada.

```
foreach ($allUsers as $notificar) {  
    User::findOrFail($notificar->id)->notify(new NotifyAdmin());  
}
```

A esta le falta un parámetro en la clase, ya que si recordamos en el constructor de la notificación recibe un arreglo.

```
public function __construct(Array $datos)  
{  
    $this->GlobalDatos = $datos;  
}
```

¿Y qué debemos enviar como parámetro?, lo que debemos pasar como parámetro es el arreglo que tiene los datos de las ventas e ingresos, es decir el “\$arregloDatos”.

```
foreach ($allUsers as $notificar) {
    User::findOrFail($notificar->id)->notify(new NotifyAdmin($arregloDatos));
}
```

Si registramos una venta podrá ver que todo funciona correctamente.

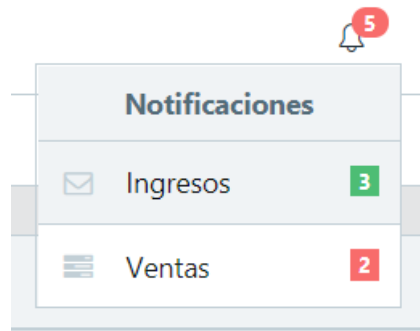
| Opciones | Artículo | Precio | Cantidad | Descuento | Subtotal |
|----------|----------|--------|----------|-----------|--------------------------------|
| | Crush | 1,80 | 2 | 0 | 3,6 |
| | | | | | Total Parcial: \$ 3.05 |
| | | | | | Total Impuesto: \$ 0.55 |
| | | | | | Total Neto: \$ 3.6 |

Vayamos a la base de datos, como puede observar tenemos 4 registros nuevos que vienen a ser los usuarios a los que les notifico. Veamos con detenimiento.

1. Tenemos el “id” que tiene un código de 36 caracteres.
2. Tenemos el “type” que viene ser de que notificador se dio.
3. Tenemos el “notifiable_id” que viene ser el código del modelo a quien se le notifico.
4. Tenemos el “notifiable_type” que viene ser el modelo al cual se le notifico.
5. Tenemos el “data” que es la información de la notificación.
6. Tenemos el “read_at” que es si la notificación ha sido leída (por defecto no está leída es decir “read_at” = null).
7. Y los campos por defecto que vienen de fecha.

| id | type | notifiable_id | notifiable_type | data | read_at | created_at | updated_at |
|--------------------------------------|-------------------------------|---------------|-----------------|---|---------|---------------------|---------------------|
| 029c355d-d4c1-4424-8d87-e82723244b15 | App\Notifications\NotifyAdmin | 9 | App\User | ["datos":["ventas":{"numero":0,"msj":"Ventas"},"in... | NULL | 2018-05-17 16:57:23 | 2018-05-17 16:57:23 |
| 1f155b64-b3a0-4140-82d5-7da3b44ce6a2 | App\Notifications\NotifyAdmin | 10 | App\User | ["datos":["ventas":{"numero":0,"msj":"Ventas"},"in... | NULL | 2018-05-17 16:57:22 | 2018-05-17 16:57:22 |
| 3a0b7a1a-af5d-46ec-bbd0-2c1a6bf19ebc | App\Notifications\NotifyAdmin | 8 | App\User | ["datos":["ventas":{"numero":0,"msj":"Ventas"},"in... | NULL | 2018-05-17 16:57:23 | 2018-05-17 16:57:23 |
| c074e658-da03-4ad1-b5f1-3bf18ec3c592 | App\Notifications\NotifyAdmin | 1 | App\User | ["datos":["ventas":{"numero":0,"msj":"Ventas"},"in... | NULL | 2018-05-17 16:57:23 | 2018-05-17 16:57:23 |

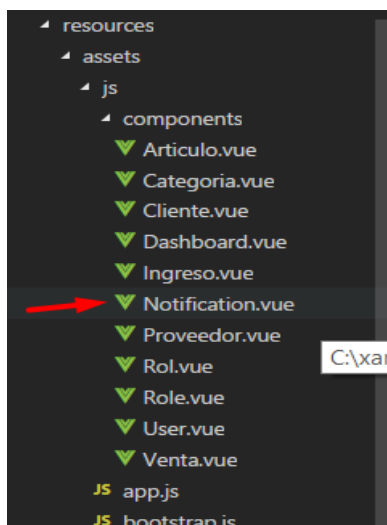
Ahora lo que necesitamos extraer es el diseño de esa notificación.



Este se encuentran dentro del directorio “resources/views/principal”.

```
<ul class="nav navbar-nav ml-auto">
  <li class="nav-item d-md-down-none">
    <a class="nav-link" href="#" data-toggle="dropdown">
      <i class="icon-bell"></i>
      <span class="badge badge-pill badge-danger">5</span>
    </a>
    <div class="dropdown-menu dropdown-menu-right">
      <div class="dropdown-header text-center">
        <strong>Notificaciones</strong>
      </div>
      <a class="dropdown-item" href="#">
        <i class="fa fa-envelope-o"></i> Ingresos
        <span class="badge badge-success">3</span>
      </a>
      <a class="dropdown-item" href="#">
        <i class="fa fa-tasks"></i> Ventas
        <span class="badge badge-danger">2</span>
      </a>
    </div>
  </li>
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle nav-link" data-toggle="
      "
    >
```

Lo cortamos y pegamos dentro de un nuevo componente que le llamaremos “Notification”



Registramos el componente dentro del “app.js”


```

Vue.component('dashboard', require('./components/Dashboard.vue'));
Vue.component('categoria', require('./components/Categoria.vue'));
Vue.component('articulo', require('./components/Articulo.vue'));
Vue.component('cliente', require('./components/Cliente.vue'));
Vue.component('proveedor', require('./components/Proveedor.vue'));
Vue.component('rol', require('./components/Rol.vue'));
Vue.component('user', require('./components/User.vue'));
Vue.component('ingreso', require('./components/Ingreso.vue'));
Vue.component('venta', require('./components/Venta.vue'));
Vue.component('notificacion', require('./components/Notification.vue'));
|
const app = new Vue({
  ...el: '#app',
  ...data: {
    ...menu: 0
  }
});

```

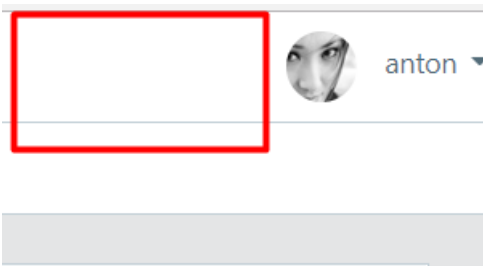
Y dentro de dicho componente pegamos el contenido que hemos cortado.

```

<template>
  <li class="nav-item d-md-down-none">
    <a class="nav-link" href="#" data-toggle="dropdown">
      <i class="icon-bell"></i>
      <span class="badge badge-pill badge-danger">5</span>
    </a>
    <div class="dropdown-menu dropdown-menu-right">
      <div class="dropdown-header text-center">
        <strong>Notificaciones</strong>
      </div>
      <a class="dropdown-item" href="#">
        <i class="fa fa-envelope-o"></i> Ingresos
        <span class="badge badge-success">3</span>
      </a>
      <a class="dropdown-item" href="#">
        <i class="fa fa-tasks"></i> Ventas
        <span class="badge badge-danger">2</span>
      </a>
    </div>
  </li>
</template>

```

Ahora una vez que tenemos listo recargamos el navegador, Como ve no hay nada porque lo que nos falta importar es el componente dentro de la plantilla "principal".



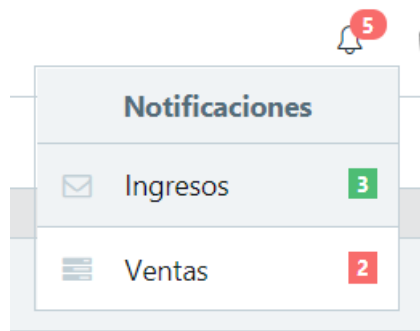
Importamos el componente que le hemos llamado “notificación”, lo puede verificar en el “app.js”, dentro del directorio “resources/assets/js/”.

```
Vue.component('dashboard', require('./components/Dashboard.vue'));
Vue.component('categoria', require('./components/Categoria.vue'));
Vue.component('articulo', require('./components/Articulo.vue'));
Vue.component('cliente', require('./components/Cliente.vue'));
Vue.component('proveedor', require('./components/Proveedor.vue'));
Vue.component('rol', require('./components/Rol.vue'));
Vue.component('user', require('./components/User.vue'));
Vue.component('ingreso', require('./components/Ingreso.vue'));
Vue.component('venta', require('./components/venta.vue'));
Vue.component('notificacion', require('./components/Notification.vue'));

const app = new Vue({
  el: '#app',
  data: {
    menu: 0
  }
});
```

Lo importamos y volvemos a recargar el navegador.

```
<ul class="nav navbar-nav ml-auto">
  <notificacion></notificacion>
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle nav-link" data-toggle="dropdown">
      
      <span class="d-md-down-none">{{Auth::user()->usuario}} </span>
    </a>
    <div class="dropdown-menu dropdown-menu-right">
      <div class="dropdown-header text-center">
        <strong>Cuenta</strong>
      </div>
      <a class="dropdown-item" href="{{ route('logout') }}">
        onclick="event.preventDefault(); document.getElementById('logout')
        <i class="fa fa-lock"></i> Cerrar sesión</a>
      <form id="logout-form" action="{{ route('logout') }}" method="PO
```



Ahora lo que haremos es ir al “app.js” y declarar una propiedad “data” que reciba un arreglo vacío que le llamaremos “notifications”

```
const app = new Vue({
  el: '#app',
  data: {
    menu: 0,
    notifications: []
  }
});
```

Lo siguiente a realizar es alimentar ese arreglo con las notificaciones que han venido agregándose, mediante peticiones “AXIOS”.

Entonces programamos una petición Axios en el método “created()” para que se ejecute cuando se crea el objeto Vue, esta petición apuntara a una ruta que aún no hemos creado.

```
created() {
  let me = this;
  axios.post('notification/get').then(function(response) {
    console.log(response.data);
  }).catch(function(error) {
    console.log(error);
  });
}
```

Ahora vayamos al archivo de rutas “web.php” y creamos dicha ruta, que apunte a un nuevo controlador que le llamaremos “NotificationController.php” y apunte en específico a su método “get”.

```
//Notificaciones
Route::post('/notification/get', 'NotificationController@get');
```

Creamos el controlador con el siguiente comando: **php artisan make:controller NotificationController**

Esta ruta debe estar fuera de todos los middleware ya que si esta en uno en específico los demás que ingresen no podrán visualizar las notificaciones.

```
Route::group(['middleware'=>['auth']],function(){
    ...
    Route::post('/logout', 'Auth\LoginController@logout')->name('logout');
    ...
    Route::get('/main', function () { ...
    })->name('main');
    ...
    //Notificaciones
    Route::post('/notification/get', 'NotificationController@get');
    ...
    //Dashboard
    Route::get('/dashboard', 'DashboardController');
    ...
    //Genrar PDF de la Venta
    Route::get('/venta/pdf/{id}', 'VentaController@pdf')->name('venta_pdf');

    Route::group(['middleware' => ['Almacenero']], function () { ...
    });

    Route::group(['middleware' => ['Vendedor']], function () { ...
    });

    Route::group(['middleware' => ['Administrador']], function () { ...
    });
});
```

Nos vamos a dicho controlador y creamos la acción “get”, y hacemos que retorne todas las notificaciones (recordar que debemos importar el modelo “Notification” para comunicarnos con la DB, es por ello que lo creamos al inicio).

```
use App\Notification;

public function get()
{
    return Notification::all();
}
```

Recargamos el navegador y presionamos click derecho en inspeccionar. La imagen a continuación es la consola del navegador y de dónde sale esta data?, esto carga del console.log que recibía la data de las notificaciones.

```
app.js:2296
▼ (4) [{...}, {...}, {...}, {...}] ⓘ
  ▼ 0:
    created_at: "2018-05-17 21:40:31"
    data: '{"datos":{"ventas":{"numero":1,"msj":"Ventas"},"ingresos":{"numero":1,"msj":"Ingresos"}}}'
    id: 0
    notifiable_id: 10
    notifiable_type: "App\User"
    read_at: null
    type: "App\Notifications\NotifyAdmin"
    updated_at: "2018-05-17 21:40:31"
    __proto__: Object
  ▼ 1:
    created_at: "2018-05-17 21:40:31"
    data: '{"datos":{"ventas":{"numero":1,"msj":"Ventas"},"ingresos":{"numero":1,"msj":"Ingresos"}}}'
    id: 5
    notifiable_id: 10
    notifiable_type: "App\User"
    read_at: null
    type: "App\Notifications\NotifyAdmin"
    updated_at: "2018-05-17 21:40:31"
    __proto__: Object
```

```

created() {
  let me = this;
  axios.post('notification/get').then(function(response) {
    console.log(response.data);
  }).catch(function(error) {
    console.log(error);
  });
}

```

Una vez que comprobamos que llegan las notificaciones de manera correcta, pasemos a alimentar el arreglo notificaciones.

```

axios.post('notification/get').then(function(response) {
  //console.log(response.data);
  me.notifications = response.data;
}).catch(function(error) {
  console.log(error);
});

```

Visualizamos en el depurador de Vue en el navegador que efectivamente el arreglo ya se almaceno la información de las notificaciones.

The screenshot shows the Vue DevTools component inspector. The top section displays the component tree with <Root> selected. The bottom section shows the 'data' property of the root component, which contains a 'menu' property with value 0 and a 'notifications' property with an array of 4 objects. The first object in the array is highlighted with a red box.

```

<Root> = $vm0
  <Notificacion>
  <Dashboard>

<Root> 🔍 Filter inspected data <>

▼ data
  menu: 0
  ▼ notifications: Array[4]
    ▼ 0: Object
      created_at: "2018-05-17 21:40:31"
      data: '{"datos":{"ventas":{"numero":1,"msj":"Ventas"},"ingresos":{"numero":0'
      id: 0
      notifiable_id: 10
      notifiable_type: "App\User"
      read_at: null
      type: "App\Notifications\NotifyAdmin"
      updated_at: "2018-05-17 21:40:31"
    ▶ 1: Object
    ▶ 2: Object
    ▶ 3: Object

```

Una vez que ya tenemos lista la información en el arreglo, nos vamos al componente “<notification>” que se encuentra dentro de la plantilla “principal.balde.php”, y ahí le vamos a pasar el arreglo de notificaciones mediante el evento “v-bind” (o simplemente “:”) para que el valor que reciba pueda ser tratado como una expresión JavaScript en este caso un arreglo), esta información “notifications” será almacenada en un evento “v-bind” que lo vincularemos con un prop que le llamaremos “notifications”.

Nota: “el atributo “prop” es un atributo personalizado usado para pasar información a un componente”

```
<notification :notifications="notifications"></notification>
```

Es decir “: **notifications**” es el prop que almacenara la información para luego pasarla al componente, este prop es alimentado del arreglo (= “**notifications**”) este arreglo es el que está siendo alimentado en la consulta con AXIOS que se encuentra en el “app.js”.

Un pequeño repaso del proceso que debe seguir.

```
const app = new Vue({
  el: '#app',
  data: {
    menu: 0,
    notifications: [] ← 2
  },
  created() {
    let me = this;
    axios.post('notification/get').then(function(response) {
      //console.log(response.data);
      me.notifications = response.data; ← 1
    }).catch(function(error) {
      console.log(error);
    });
  }
});
```

```
<strong>Notificaciones</strong>
</div>
<notification :notification="notifications"></notification>
div>
```

Y por último se almacena en el prop “notification”

```
</div>
<notification :notification="notifications"></notification>
div>
```

Una vez que la información esta almacenada en el prop, este puede ser utilizado dentro del componente, ¿de qué manera?, mediante el atributo “**props**”. ¿Y qué información va a pasar?, la información que le va pasar al componente son el arreglo de notificaciones.


Programemos el prop dentro del componente.

Como ven simplemente declaramos el atributo “props” (antes de declarar la propiedad “data ()”) y en un arreglo escribimos la propiedad prop que espera escribir (en este caso “notifications”).

```
export default {  
  props : ['notifications'],  
  data () {  
    return {  
  
    }  
  }  
}
```

Nota: Recuerde que el nombre de esa propiedad es la que recibe del componente (Eje: Si la propiedad se llama “: recibir” en el prop también debe llamarse así).

```
</div>  
<notificacion :notifications="notifications"></notificacion>  
iv>
```



Ahora una vez que el prop ya tiene la información, la recorreremos con una directiva “v-for”.

Entonces empecemos, agregaremos un antes de abrir el <a> que muestra el ingreso y su cantidad.

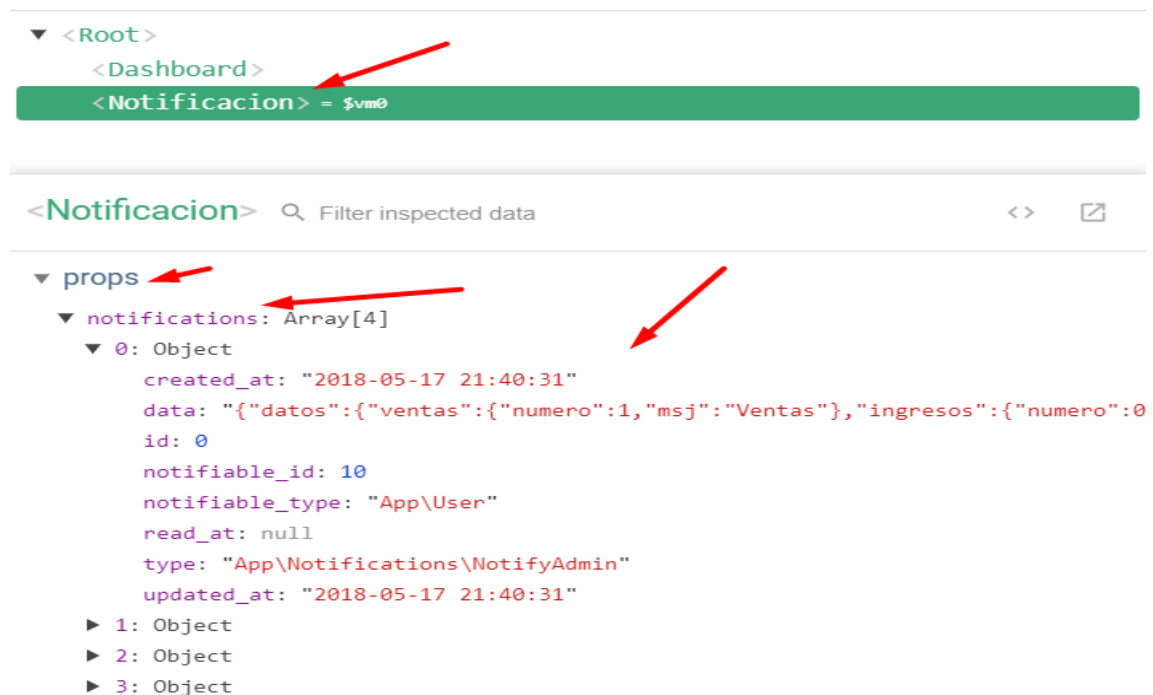
```
<div class="dropdown-menu dropdown-menu-right">  
  <div class="dropdown-header text-center">  
    <strong>Notificaciones</strong>  
  </div>  
  <li v-for="item in notifications" :key="item.id">  
    <a class="dropdown-item" href="#">  
      <i class="fa fa-envelope-o"></i> Ingresos  
      <span class="badge badge-success">3</span>  
    </a>  
    <a class="dropdown-item" href="#">  
      <i class="fa fa-tasks"></i> Ventas  
      <span class="badge badge-danger">2</span>  
    </a>  
  </li>  
</div>
```

La directiva “v-for” es algo que ya hemos explicado antes con detenimiento así que no nos detendremos en esto.

```
<li v-for="item in notifications" :key="item.id">
```

Nota: lo único que aclaramos es que “notifications” es el prop que hemos capturado y lo trabaja como si fuese un arreglo más, es por ello que lo puedo recorrer con un “vfor”

Ahora si revisamos el depurador de Vue e ingresamos al componente “Notification” podrá ver que aunque no hayamos declarado ninguna variable (en este caso arreglo) en la propiedad “data()”, este interpreta como a “notifications” de prop como si fuese una variable más, ver figura.



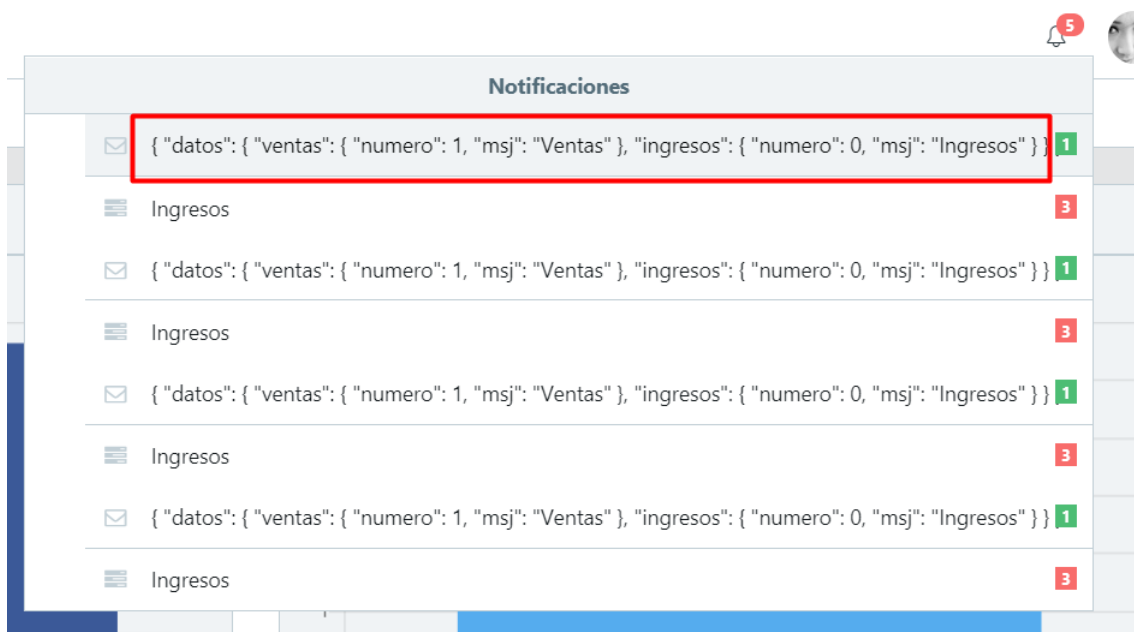
Ahora como pueden ver para ingresar a la información que necesitamos, debemos ingresar al atributo “data” del objeto, así que vayamos escribiendo dicho código.

```
<i class="fa fa-envelope-o"></i> {{ item.data }}
```

Pero como la data que llega es una cadena esto debe convertirse en un objeto de JavaScript, para hacer ello utilizaremos “JSON.parse(item.data)”.

```
<i class="fa fa-envelope-o"></i> {{ JSON.parse(item.data) }}
```

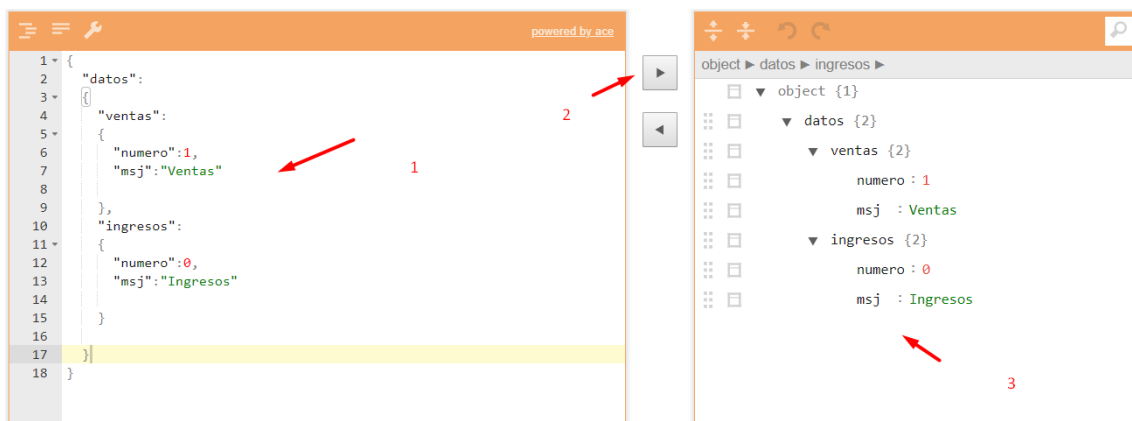
Recargamos el navegador y podrá ver lo que se visualiza en la notificación.



Como pueden ver es un objeto, si aún no pueden distinguirlo bien, copiamos el contenido de la data, para ello ingresamos al depurador de Vue y copiamos la data.

```
▼ notifications: Array[4]
  ▼ 0: Object
    created_at: "2018-05-17 21:40:31"
    data: [{"datos":{"ventas":{"numero":1,"msj":"Ventas"},"ingresos":{"numero":0,"msj":"Ingresos"}}]
    id: 0
    notifiable_id: 10
    notifiable_type: "App\User"
```

Vayamos a la siguiente URL: <https://jsoneditoronline.org/> ahí pegamos la data, simplemente le damos forma presionando enter y hacemos click en la fecha que apunta a la derecha.

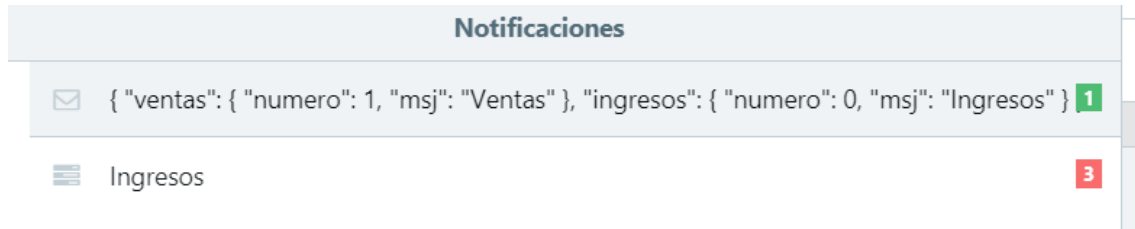


Ahí pueden ver de manera más clara a donde debemos ir ingresando para llegar a la data que nos interesa.

Entonces todo eso está dentro de data, el primer objeto donde debemos ingresar es a “datos”

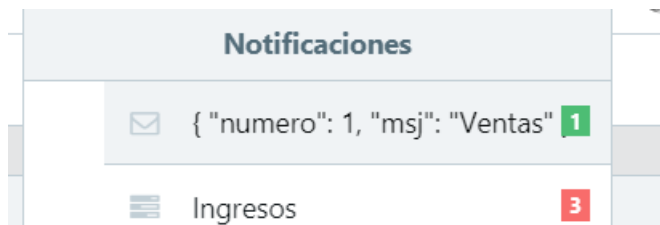
Entonces después de cerrar el paréntesis seguido presionamos “.” Y luego escribimos “datos”.

```
<i class="fa fa-envelope-o"></i> {{ JSON.parse(item.data).datos }}
```



Luego ingresamos a ventas.

```
{{ JSON.parse(item.data).datos.ventas }}
```

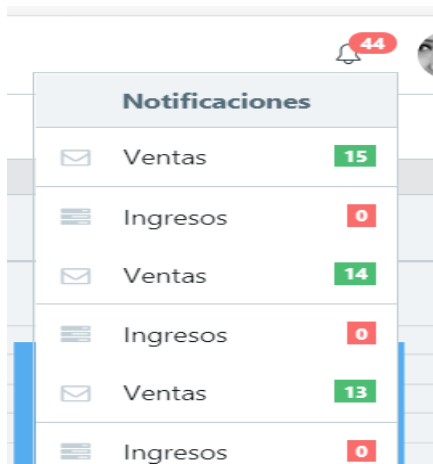


Y por último al dato que queremos ingresar que en este caso ahí debe ir el mensaje que es “ventas”.

Hagamos lo mismo con el número de ventas y el resto de ingresos.

```
<li v-for="item in notifications" :key="item.id">
  <a class="dropdown-item" href="#">
    <i class="fa fa-envelope-o"></i> {{ JSON.parse(item.data).datos.ventas.msj }}
    <span class="badge badge-success">{{ JSON.parse(item.data).datos.ventas.numero }}</span>
  </a>
  <a class="dropdown-item" href="#">
    <i class="fa fa-tasks"></i> {{ JSON.parse(item.data).datos.ingresos.msj }}
    <span class="badge badge-danger">{{ JSON.parse(item.data).datos.ingresos.numero }}</span>
  </a>
</li>
```

Si recargamos el navegador podrá ver que esta la venta que realizamos anteriormente y como no hemos realizado ninguna compra o ingreso está vacío.



Podemos además agregar el número de notificaciones en la parte superior de las notificaciones (para que sea dinámico en base a la nueva notificación) entonces cambiémoslo.

```
{{ notifications.length }}
```

Se debe agregar en esta parte de la notificación.

```
<a class="nav-link" href="#" data-toggle="dropdown">
  <i class="icon-bell"></i>
  <span class="badge badge-pill badge-danger"> {{ notifications.length }} </span>
</a>
```

Sin embargo las notificaciones que se almacenan en la base de datos, necesitan una forma conveniente de acceder a ellas (para evitar parsearlas cada vez que las requerimos) desde sus entidades *notifiables*. El *trait* Illuminate\Notifications\Notifiable, que se incluye en el modelo predeterminado de App\User, incluye una relación Eloquent con notifications que devuelve las notificaciones de la entidad. Para obtener notificaciones, puede acceder a este método como cualquier otra relación Eloquent. De forma predeterminada, las notificaciones se ordenarán por el campo de fecha created_at.

Entonces vamos al controlador "NotificationController" e implementaremos el facade Auth para mostrar las notificaciones del usuario logeado.

```
use Auth;
```

Luego en el método "get" modificaremos un poco y usaremos el facade "Auth::user()" (esto capturara el id del usuario logeado) y como dijimos anteriormente el trait notifiables tiene una relación Eloquent con las notificaciones así que ingresamos a la relación Eloquent.

```
Auth::user()->notifications
```

Y lo finalmente lo retornamos

```
return Auth::user()->notifications;
```

El controlador quedaría de la siguiente manera.

```

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Notification;
use Auth;

class NotificationController extends Controller
{
    ... public function get()
    ... {
    ...     //Obtener todas las notificaciones del usuario logeado con sus datos
    ...     return Auth::user()->notifications;
    ... }
}

```

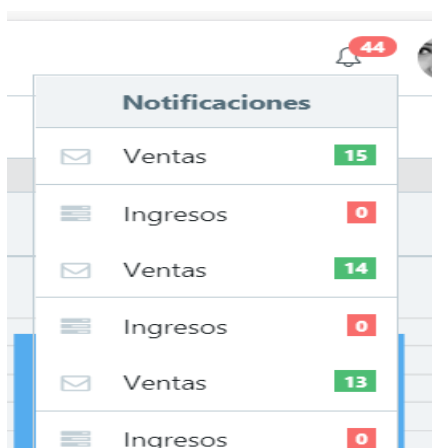
Ahora antes de recargar el navegador en el componente quitamos el “JSON.parse()” como hicimos anteriormente.

```

<li v-for="item in notifications" :key="item.id">
  <a class="dropdown-item" href="#">
    <i class="fa fa-envelope-o"></i> {{ item.data.datos.ventas.msj }}
    <span class="badge badge-success">{{ item.data.datos.ventas.numero }}</span>
  </a>
  <a class="dropdown-item" href="#">
    <i class="fa fa-tasks"></i> {{ item.data.datos.ingresos.msj }}
    <span class="badge badge-danger">{{ item.data.datos.ingresos.numero }}</span>
  </a>
</li>

```

Ahora si recarguemos el navegador.



Como ve las notificaciones se leen perfectamente, ya que para el ejemplo he realizado 02 ventas el día de hoy y ahí se ven (aun no es en tiempo real ya que después de hacer una venta tengo que recargar el navegador, para solucionar ello utilizaremos más adelante “Pusher” y “broadcast”) sin embargo arriba salen 03 notificaciones y eso es un error, ¿Por qué? Porque lo que necesito son las ventas del día y simplemente me está trayendo todas las notificaciones que

tiene dicho usuario y al día anterior ya había hecho una venta es por ello que me muestra 03. Para solucionar ello debemos verificar que la notificación es de hoy de lo contrario marcarla como leída.

Laravel también facilita ese proceso, entonces Si desea recuperar sólo las notificaciones "no leídas", puede utilizar la relación "unreadNotifications". De nuevo, estas notificaciones se clasificarán por el campo de fecha created_at:

Entonces lo primero que debemos realizar es obtener un listado de las notificaciones no leídas del usuario autenticado, para ello utilizaremos "unreadNotifications"

```
Auth::user()->unreadNotifications
```

Luego de obtener el listado de notificaciones del usuario autenticado, necesitamos recorrer las notificaciones para saber cuál de ellas no es del día actual para marcarla como leída.

```
foreach ($unreadNotifications as $notification) {  
    //code  
}
```

Entonces realizamos la verificación, compararemos la fecha actual con la fecha de la notificación, si no son iguales la marcaremos como leída de lo contrario mantendrá su estado.

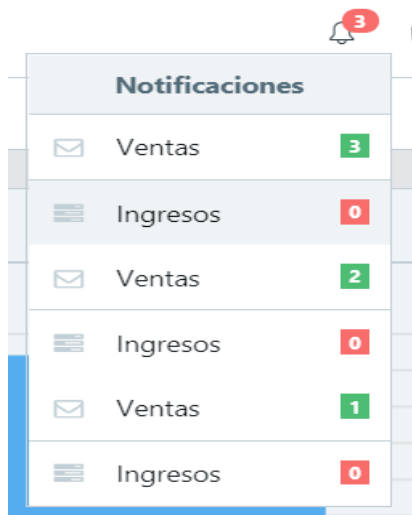
Nota: El trait Illuminate\Notifications\Notifiables proporciona un método markAsRead, que actualiza la columna read_at en el registro de la base de datos de notificación.

```
foreach ($unreadNotifications as $notification) {  
    $fechaActual= date('Y-m-d');  
    if ($fechaActual != $notification->created_at->toDateString()) {  
        $notification->markAsRead();  
    }  
}
```

Y como ya no queremos todas las notificaciones, sino solo las leídas cambiamos el "notifications" por "unreadNotifications".

```
return Auth::user()->unreadNotifications;
```

Recargamos el navegador y podrá apreciar que ya está solucionado ese pequeño inconveniente.



Podemos mejorar un poco la presentación haciendo una pequeña validación para verificar si existen o no notificaciones mediante la directiva “v-if – v-else”, es decir si (v-if) existen notificaciones que me las liste, de lo contrario mostrar un mensaje diciendo que no tiene modificaciones.

```
<div v-if="notifications.length">
  <li v-for="item in notifications" :key="item.id">
    <a class="dropdown-item" href="#">
      <i class="fa fa-envelope-o"></i> {{ item.data.datos.ventas.msj }}
      <span class="badge badge-success">{{ item.data.datos.ventas.numero }}</span>
    </a>
    <a class="dropdown-item" href="#">
      <i class="fa fa-tasks"></i> {{ item.data.datos.ingresos.msj }}
      <span class="badge badge-danger">{{ item.data.datos.ingresos.numero }}</span>
    </a>
  </li>
</div>
<div v-else>
  <a><span>No tiene notificaciones</span></a>
</div>
```

Como existen notificaciones no se mostrara el “v-else” pero esto se visualizara si no existen notificaciones.

Lo próximo a realizar es empezar a trabajar en la transmisión de eventos, ahora nos ubicamos en nuestro archivo “.env”, aquí debemos agregar el drive que utilizaremos para hacer la transmisión de eventos, el cual será “Pusher” podemos verificar el nombre del drive en el archivo “Broadcasting.php” dentro del directorio “config/”, ahí podrá ver el nombre del drive.

```

... 'connections' => [
    ... 'pusher' => [
        ... 'driver' => 'pusher',
        ... 'key' => env('PUSHER_APP_KEY'),
        ... 'secret' => env('PUSHER_APP_SECRET'),
        ... 'app_id' => env('PUSHER_APP_ID'),
        ... 'options' => [
            ... 'cluster' => env('PUSHER_APP_CLUSTER'),
            ... 'encrypted' => true,
        ],
    ],
    ... 'redis' => [
        ... 'driver' => 'redis',
        ... 'connection' => 'default',
    ],
    ... 'log' => [
        ... 'driver' => 'log',
    ],
    ... 'null' => [
        ... 'driver' => 'null',
    ],
],
];

```

Así actualizamos nuestro BROADCAST_DRIVER del archivo .env, por defecto viene con “log”.

```
BROADCAST_DRIVER=pusher
```

Luego verificar que dentro del “Broadcasting.php” del directorio “config/”, tiene la variable “encrypted” con valor “true”.

Ahora nos vamos a nuestro archivo de notificación “NotifyAdmin” (app\Notifications) y agregaremos un nuevo canal de entrega, el cual será “broadcast”.

```
return ['database', 'broadcast'];
```

Como el nuevo canal de entrega será mediante “broadcast” necesitamos un nuevo medio para enviarlo así que lo creamos.

```

public function toBroadcast($notifiable)
{
    return [
        'datos' => $this->GlobalDatos
    ];
}

```

```

class NotifyAdmin extends Notification
{
    use Queueable;

    public $GlobalDatos;

    /** ...
    public function __construct(Array $datos)
    {
        $this->GlobalDatos = $datos;
    }

    /** ...
    public function via($notifiable)
    {
        return ['database', 'broadcast'];
    }

    /** ...
    public function toDatabase($notifiable)
    {
        return [
            'datos' => $this->GlobalDatos
        ];
    }

    public function toBroadcast($notifiable)
    {
        return [
            'datos' => $this->GlobalDatos
        ];
    }
}

```

Para que el usuario autenticado escuche los eventos que suceden (Eje. Un usuario “X” registra una nueva venta y de manera automática se le notifique lo que sucede al usuario que le corresponde escuchar en la aplicación), este usuario debe estar autorizado para poder escuchar este canal privado (a través del canal se envía la información), entonces ¿en donde definimos las reglas de autorización del canal?, es en el archivo “channels” del directorio “routes”.

Por defecto en este archivo hay una regla de autorización del modelo usuario, el cual verifica si el “id” del usuario autenticado es igual al “id” del usuario notificado.


```

/*
|-----
| Broadcast Channels
|-----
|
| Aquí puede registrar todos los canales de transmisión de eventos que su
| aplicación sea compatible. Las devoluciones de llamada de autorización del canal son
| utilizadas para verificar si un usuario autenticado puede escuchar el canal.
|
|
*/

Broadcast::channel('App.User.{id}', function ($user, $id) {
    ... return (int) $user->id === (int) $id;
});

```

Ahora programemos el canal del lado del cliente en el archivo app.js, dentro del método created() que es el que se ejecuta al crearse el objeto Vue.

Enlace: <https://laravel.com/docs/5.6/broadcasting#notifications>

Al combinar la transmisión de eventos (Broadcasting) con notificaciones (notifications), su aplicación JavaScript puede recibir nuevas notificaciones a medida que ocurren, sin necesidad de actualizar la página.

Una vez que haya configurado una regla de autorización para usar el canal de transmisión (es decir en el archivo “channels.php”), puede escuchar los eventos de transmisión usando el método **“notification”** de “Echo”. Recuerde, el nombre del canal debe coincidir con el nombre de la clase de la entidad que recibe las notificaciones:

```

Echo.private('App.User.' + userId).notification((notification) => {
    console.log(notification);
});

```

En este ejemplo, todas las notificaciones enviadas a instancias a través del canal serían recibidas por la devolución de llamada. La devolución de llamada de autorización de canal para el canal se incluye en el valor predeterminado que se envía con el marco de Laravel.

Para obtener el “id” del usuario autenticado y verificar si dicho usuario puede escuchar en dicho canal, necesitamos de alguna manera capturar el “id” del usuario autenticado para ello capturaremos el contenido en una meta etiqueta que tendrá el “id” del usuario autenticado.

```

var userId = $('meta[name="userId"]').attr('content');

```

Si recargamos el navegador nos aparecerá un error diciendo que no está definido el selector “\$”.

```
s/Chart.js/2.7.2/Chart.min.js".
[Vue warn]: Error in created hook: "ReferenceError: $ is not defined"
    (found in <Root>)
ReferenceError: $ is not defined
    at Vue$.created (app.js:2304)
    at callHook (app.js:30653)
    at Vue$.Vue._init (app.js:32318)
    at new Vue$3 (app.js:32417)
    at Object.defineProperty.value (app.js:2289)
    at __webpack_require__ (app.js:20)
    at Object.<anonymous> (app.js:2253)
    at __webpack_require__ (app.js:20)
    at Object.defineProperty.value (app.js:63)
    at app.js:66
You are running Vue in development mode      app is 36207
```

Esto es debido a que estamos utilizando el selector “\$” (dólar) de JQuery, sin embargo no hemos importado JQuery antes de usarlo es por ello que salta la excepción, por lo que si nos fijamos en la parte superior (al inicio, primera línea) del archivo “app.js” requerimos el archivo “Bootstrap”

```
require('./bootstrap');
```

Este archivo no se debe confundir con el framework Bootstrap del frontend para el maquetado, sino este archivo contiene diferentes bibliotecas requeridas para nuestro proyecto.

Como podrá ver en la línea 10 tenemos comentado la línea de JQuery.

```
//window.$ = window.jQuery = require('jquery');
```

Lo que deberíamos realizar es descomentarla (no olvidar tener levantado “npm run watch” de lo contrario levantarlo de nuevo o utilizar “npm run dev”).

```
const app = new Vue({
  el: '#app',
  data: {
    menu: 0,
    notifications: []
  },
  created() {
    let me = this;
    axios.post('notification/get').then(function(response) {
      //console.log(response.data);
      me.notifications = response.data;
    }).catch(function(error) {
      console.log(error);
    });
  },
  var userId = $('meta[name="userId"]').attr('content');
  Echo.private('App.User.' + userId).notification((notification) => {
    console.log(notification);
  });
});
```

Entonces volvamos a la meta etiqueta, ¿Pero dónde se encuentra esa meta etiqueta?, esa meta etiqueta debemos crearla en el archivo principal de la plantilla por ejemplo (principal.blade.php).

Y lo que haremos es verificar si el usuario esta autenticado, si lo está que me obtenga el “id” del usuario, de lo contrario que me traiga vacío.

```
<!-- Id for Channel Notification -->
<meta name="userId" content="{ { Auth::check() ? Auth::user()->id : '' } }">
```

```
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="Sistema Ventas Laravel Vue Js- IncanatoIT">
    <meta name="author" content="Incanatoit.com">
    <meta name="keyword" content="Sistema ventas Laravel Vue Js, Sistema compras Laravel Vue Js">
    <link rel="shortcut icon" href="img/favicon.png">
    <!-- Id for Channel Notification -->
    <meta name="userId" content="{ { Auth::check() ? Auth::user()->id : '' } }">

    <title>Sistema Ventas - IncanatoIT</title>
    <meta name="csrf-token" content="{ { csrf_token() } }">
    <!-- Load Chart Js -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.2/Chart.min.js"></script>
    <!-- Icons -->
    <link href="css/plantilla.css" rel="stylesheet">
</head>
```

Antes de probar necesitamos habilitar un “Service Provider”, entonces nos vamos al directorio “config” en el archivo “app.php” y nos dirigimos a los proveedores y des comentamos:

```
// App\Providers\BroadcastServiceProvider::class,
```

Y probamos realizando una venta, como ven en la consola la información está llegando correctamente. Ahora lo que debemos hacer es ir al archivo de notificación y ponerlo todo dentro de un arreglo “data” para que tenga la misma estructura que la notificación de la DB.

```
public function toBroadcast($notifiable)
{
    return [
        'data' => [
            'datos' => $this->GlobalDatos
        ]
    ];
}
```

Ahora nos falta programar en el “app.js”, ahí lo que nos queda por realizar es alimentar el arreglo de notificaciones (notifications), es decir cada vez que haya una nueva venta se le notifique y se

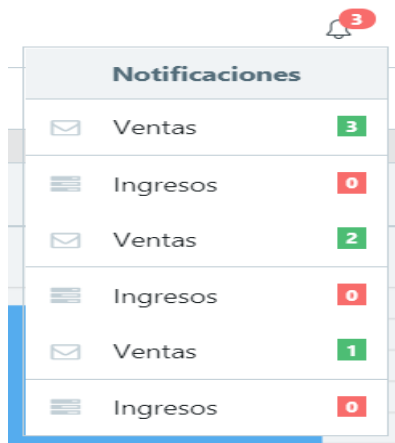
agregue al arreglo de notificaciones, para ello lo que haremos es utilizar el método “unshift” de JavaScript para que todo evento se vaya agregando al inicio del arreglo “notifications”.

```
me.notifications.unshift(notification);
```

Código completo.

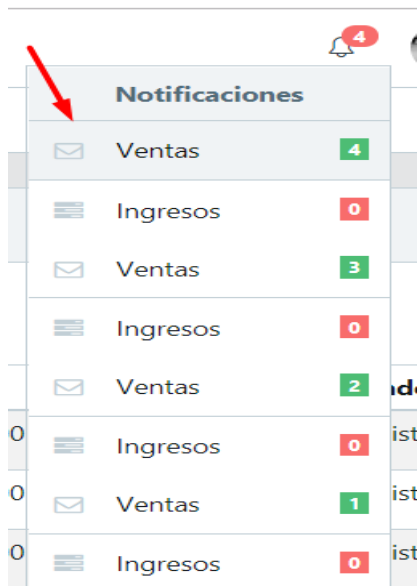
```
var userId = $('meta[name="userId"]').attr('content');  
  
Echo.private('App.User.' + userId).notification((notification) => {  
    me.notifications.unshift(notification);  
});
```

Si cargamos el navegador podrá visualizar que hay un listado de notificaciones de ventas e ingresos, es decir una detrás de otra (la lógica normal de una notificación), sin embargo nosotros no queremos eso queremos algo un poco diferente, lo que queremos es que aumente el contador de ventas mas no tener una lista de notificaciones, ver imagen.



| Notificaciones | | |
|----------------|----------|---|
| ✉ | Ventas | 3 |
| ☰ | Ingresos | 0 |
| ✉ | Ventas | 2 |
| ☰ | Ingresos | 0 |
| ✉ | Ventas | 1 |
| ☰ | Ingresos | 0 |

Si realizamos otra venta se agrega a la lista de notificaciones.



| Notificaciones | | |
|----------------|----------|---|
| ✉ | Ventas | 4 |
| ☰ | Ingresos | 0 |
| ✉ | Ventas | 3 |
| ☰ | Ingresos | 0 |
| ✉ | Ventas | 2 |
| ☰ | Ingresos | 0 |
| ✉ | Ventas | 1 |
| ☰ | Ingresos | 0 |

Entonces vamos al componente “Notifications.vue” y utilizaremos una propiedad computada, ¿para qué nos sirve?, como explicamos anteriormente las propiedades computadas son aquellas que están atentas a algún cambio y realiza algo como respuesta.

Una propiedad computada la pondremos luego de la propiedad “data()”

```
computed: {  
}
```

Entonces empecemos a crear nuestra primera propiedad computada, la cual le llamaremos “listar()”

```
computed: {  
  listar: function(){  
  }  
}
```

Ahora lo que queremos es mostrar en la notificación no son todas las notificaciones, sino la última notificación realizada.

Es por ello que buscamos de alguna manera de acceder solo a la última notificación que es la que nos importa realmente, para solucionar eso, accederemos al índice 0 del arreglo de notificaciones (recordar que el prop “notifications” es interpretado como un arreglo mas).

Lo depuramos con un “return” para verificar que está reaccionando cuando hay una nueva venta.

```
computed: {  
  listar: function(){  
    return this.notifications[0];  
  }  
}
```

Ingresamos a nuestro navegador y realizamos una nueva venta. Si todo marcha bien la propiedad computada debe enterarse del cambio en el arreglo de notificaciones y capturar el último índice del arreglo de notificaciones.



Como puede ver existen 8 notificaciones en total (prop “notifications”), sin embargo la propiedad computada me captura solo la última notificación es decir la venta número 8.

Ahora como vemos funciona bien, lo siguiente a realizar es almacenar dicho valor; así como está la propiedad computada “listar” no se puede recorrer ya que es un objeto.

Entonces empecemos a programar para poder recorrerlo.

Lo primero que haremos es... En nuestra propiedad “data” declaramos un arreglo vacío que le llamaremos “arrayNotifications”

```
data () {  
  return {  
    arrayNotifications: []  
  }  
},
```

El arreglo “arrayNotification” almacenara la última notificación, para ello utilizaremos el método “Object.values()”, podemos ver más de este método en la siguiente URL: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Object/values

El método “Object.values()” devuelve un array que contiene las propiedades **enumerables** de un objeto dado.

Entonces el método “Object” recibe un parámetro que es el arreglo “notifications”, este lo almacenamos en el arreglo “arrayNotifications”.

```
listar: function(){
  //Capturo la ultima notificación agregada
  this.arrayNotifications = Object.values(this.notifications[0]);
}
```


Recargamos el navegador y realizamos otra venta.

```
▼ arrayNotifications: Array[3]
  ► 0: Object
    1: "8c42b995-e3d9-4a99-96ee-ba12d329a9ec"
    2: "App\Notifications\NotifyAdmin"
```

▼ computed

Como ven esto es lo que retorna la notificación cuando es enviada por el canal privado.

```
▼ data
  ▼ arrayNotifications: Array[3]
    ▼ 0: Object
      ▼ datos: Object
        ▼ ingresos: Object
          msj: "Ingresos"
          numero: 0
        ▼ ventas: Object
          msj: "Ventas"
          numero: 9
      1: "8c42b995-e3d9-4a99-96ee-ba12d329a9ec"
      2: "App\Notifications\NotifyAdmin"
```



▼ computed

Como ven las propiedades del objeto las convierte en una colección de objetos que conformaran un nuevo arreglo (es decir el arreglo en el que se almacena = "arrayNotifications").

Ahora queremos acceder en donde se encuentra lo que necesitamos que son las ventas e ingresos, entonces si observamos el nuevo arreglo "arrayNotifications" en la posición "0" se encuentra los datos que necesitamos sin embargo el objeto en la posición "0" está conformada por otro objeto.

Entonces lo que haremos es volver pasarle un "object values" pero solamente a la posición "0" que es lo que nos interesa convertir en otro arreglo.

```
listar: function(){
  //Capturo la ultima notificación agregada
  this.arrayNotifications = Object.values(this.notifications[0]);
  return Object.values(this.arrayNotifications[0]);
}
```

Probamos agregando una nueva venta.



“Como ve en la línea 41 de nuestro código lo primero que hacemos es capturar el objeto que llega el cual es la última notificación, es por ello que el “arrayNotifications” además de los datos de las ventas e ingresos también tiene el “id” de la notificación y la clase notificación.

```
this.arrayNotifications = Object.values(this.notifications[0]);
```

```
▼ arrayNotifications: Array[3]
  ► 0: Object
    1: "384dbaff-f2fd-468d-b979-87c7edb5d4ec"
    2: "App\Notifications\NotifyAdmin"
```

Sin embargo si usted se da cuenta en la línea 42, solo capturamos el índice “0” del arreglo “arrayNotifications” es por eso que la propiedad computada retorna solo los datos de interés es decir las ventas e ingresos”.

```
return Object.values(this.arrayNotifications[0]);
```

```
▼ listar: Array[1]
  ▼ 0: Object
    ▼ ingresos: Object
      msj: "Ingresos"
      numero: 0
    ▼ ventas: Object
      msj: "Ventas"
      numero: 3
```


Como ven ya tenemos limpio los ingresos y las ventas, ahora lo que haremos es recorrerlos, vamos al HTML de nuestro componente y cambiemos el “notifications” por “listar” ahora podemos acceder al objeto ingresos y ventas, mediante un “punto (.)”.

```
<div v-if="listar.length">
  <li v-for="item in listar" :key="item.id">
    <a class="dropdown-item" href="#">
      <i class="fa fa-envelope-o"></i> {{ item.ventas.msj }}
      <span class="badge badge-success">{{ item.ventas.numero }}</span>
    </a>
    <a class="dropdown-item" href="#">
      <i class="fa fa-tasks"></i> {{ item.ingresos.msj }}
      <span class="badge badge-danger">{{ item.ingresos.numero }}</span>
    </a>
  </li>
</div>
```

Esto debería funcionar si recargamos el navegador sin embargo si recargamos el navegador, nos aparecerá una serie de errores.

```
✖ [Vue warn]: Error in render: "TypeError: Cannot
convert undefined or null to object" app.js:38712
found in
---> <Notificacion> at
resources\assets\js\components\Notification.vue
<Root>
✖ TypeError: Cannot convert undefined or null to
object app.js:39849
    at Function.values (<anonymous>)
    at VueComponent.listar (app.js:64538)
    at Watcher.get (app.js:41247)
    at Watcher.evaluate (app.js:41354)
    at VueComponent.computedGetter [as listar] (app.js:41606)
    at Object.get (app.js:40052)
    at Proxy.render (app.js:64571)
    at VueComponent.Vue._render (app.js:42606)
    at VueComponent.updateComponent (app.js:40897)
    at Watcher.get (app.js:41247)
```

Esto se debe a que se lanza una excepción cuando queremos recorrer en un array un objeto que no existe, como son las ventas e ingresos y usted se preguntara ¿Por qué no se puede ingresar a ellos, si hace un momento funcionaba? Y lo que sucede es que el array “arrayNotifications” recibía tres propiedades del objeto que los convertía en arreglo.

```
▼ arrayNotifications: Array[3]
  ► 0: Object
    1: "8c42b995-e3d9-4a99-96ee-ba12d329a9ec"
    2: "App\Notifications\NotifyAdmin"
```

▼ computed

De los cuales ingresaba a la posición “0” de dicho arreglo para luego recorrerlos.

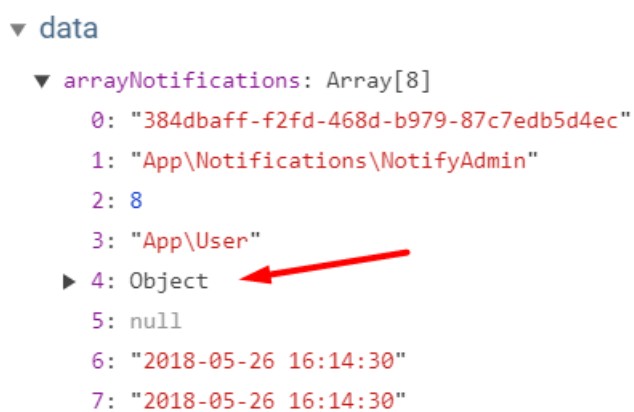
```
return Object.values(this.arrayNotifications[0]);
```

Es decir es decir a esto.



Nota: Esto sucede cuando los datos que recibe son desde el canal privado (es decir cuando ocurre una notificación).

Sin embargo si usted se va ahora mismo al depurador de Vue, lo que acaba de recibir es algo diferente.



Esto es porque las notificaciones que cargan no son de un canal privado es decir desde aquí.

```

const app = new Vue({
  el: '#app',
  data: {
    menu: 0,
    notifications: []
  },
  created() {
    let me = this;
    axios.post('notification/get').then(function(response) {
      //console.log(response.data);
      me.notifications = response.data;
    }).catch(function(error) {
      console.log(error);
    });

    var userId = $('meta[name="userId"]').attr('content');

    //**
    Echo.private('App.User.' + userId).notification((notification) => {
      me.notifications.unshift(notification);
    });
  }
});

```

Sino los datos son cargados desde la consulta de Axios.

```

const app = new Vue({
  el: '#app',
  data: {
    menu: 0,
    notifications: []
  },
  created() {
    let me = this;
    axios.post('notification/get').then(function(response) {
      //console.log(response.data);
      me.notifications = response.data;
    }).catch(function(error) {
      console.log(error);
    });

    var userId = $('meta[name="userId"]').attr('content');

    //**
    Echo.private('App.User.' + userId).notification((notification) => {
      me.notifications.unshift(notification);
    });
  }
});

```

Es por ello que cargan más propiedades. Entonces lo que haremos es hacer una condicional donde diremos si el tamaño del arreglo “arrayNotifications” es mayor a 3 es porque viene desde una consulta AXIOS, por lo tanto debemos capturar los datos en el índice “4” de lo contrario los datos vienen desde el canal privado por lo tanto es capturado los datos desde el índice “0”.

```

//Validación por índice fuera de rango
if (this.arrayNotifications.length > 3) {
  //Si el tamaño es > 3 Es cuando las notificaciones son obtenidas desde el mismo servidor, es
  decir por la consulta con AXIOS
  return Object.values(this.arrayNotifications[4]);
} else {
  //Si el tamaño es < 3 Es cuando las notificaciones son obtenidas desde el canal privado, es
  decir mediante Laravel Echo y Pusher

```

```
return Object.values(this.arrayNotifications[0]);
}
```

Ahora si recarguemos el navegador.

The screenshot shows a web application interface on the left and the Redux DevTools state on the right. The web application has a header with a notification bell icon and the name 'anton'. Below the header is a 'Notificaciones' (Notifications) section with two items: 'Ventas' (Sales) with a green badge showing '3' and 'Ingresos' (Income) with a red badge showing '0'. The main content area shows a blue bar chart for 'Ventas' (Sales) for the month of 'Mayo' (May). Below the chart, it says 'Realizadas por mes en el año 2018' (Completed by month in the year 2018). At the bottom, it says 'Desarrollado por IncanatoIT' (Developed by IncanatoIT).

The Redux DevTools state on the right shows the following structure:

- <Root>**
 - <Notificacion> = \$vm0**
 - <Dashboard>**
- <Notificacion>**
 - props**
 - notifications:** Array[3]
 - data**
 - arrayNotifications:** Array[8]
 - 0: "384dbaff-f2fd-468d-b979-87c7edb5d4ec"
 - 1: "App\Notifications\NotifyAdmin"
 - 2: 8
 - 3: "App\User"
 - 4: Object
 - datos:** Object
 - 5: null
 - 6: "2018-05-26 16:14:30"
 - 7: "2018-05-26 16:14:30"
 - computed**
 - listar:** Array[1]
 - 0: Object
 - ingresos:** Object
 - ventas:** Object

Ahora podrá ver que funciona correctamente sin embargo persiste el error del objeto vacío eso es porque en el arreglo "arrayNotifications" en la propiedad "read_at" recibe null porque no está leída, pero es nada que nos debamos preocupar porque funciona todo correctamente.

Ahora que pasa si ingresamos al siguiente día y no existen notificaciones, entonces hagamos una prueba borrando las notificaciones de la base de datos.

The screenshot shows a database management tool interface with a list of tables on the left. The tables are: ingresos, migrations, notifications, password_resets, personas, proveedores, roles, users, and ventas. The 'notifications' table is highlighted with a red arrow. A confirmation dialog is open in the center, asking for confirmation to truncate the 'notifications' table. The dialog text is: '¡Está a punto de TRUNCAR una tabla completa! ¿Realmente desea ejecutar "TRUNCATE `notifications`"?'. There is a checkbox labeled 'Habilite la revisión de las claves foráneas' (Enable foreign key checks) which is checked. The dialog has 'OK' and 'Cancelar' (Cancel) buttons.

Y recargamos el navegador.

```
✖ [Vue warn]: Error in render: "TypeError: Cannot  
convert undefined or null to object" app.js:38712  
  
found in  
  
---> <Notificacion> at  
resources/assets/js/components/Notification.vue  
  <Root>  
  
✖ TypeError: Cannot convert undefined or null to  
object app.js:39849  
  at Function.values (<anonymous>)  
  at VueComponent.listar (app.js:64538)  
  at Watcher.get (app.js:41247)  
  at Watcher.evaluate (app.js:41354)  
  at VueComponent.computedGetter [as listar] (app.js:41606)  
  at Object.get (app.js:40052)  
  at Proxy.render (app.js:64571)  
  at VueComponent.Vue._render (app.js:42606)  
  at VueComponent.updateComponent (app.js:40897)  
  at Watcher.get (app.js:41247)
```

Nos saldrá dicho error pero ahora otra vez por arreglo fuera de índice, y esto es debido a que como no llegan notificaciones, el "Object.values" recibe la última notificación, sin embargo como pueden ver no existe notificaciones es por ello que sale el error .

```
▼ props  
  ▼ notifications: Array[0]  
  
▼ data  
  ▼ arrayNotifications: Array[0]  
  
▼ computed  
  listar: "(error during evaluation)"
```

Solucionemos eso simplemente haciendo una validación, verificando si existen o no notificaciones en el "prop".

```
if (this.notifications == '') {  
  return this.arrayNotifications = [];  
} else {  
  //Capturo la ultima notificación agregada  
  this.arrayNotifications = Object.values(this.notifications[0]);  
  //Validación por índice fuera de rango  
  if (this.arrayNotifications.length > 3) {  
    //Si el tamaño es > 3 Es cuando las notificaciones son obtenidas desde el mismo servidor,  
    es decir por la consulta con AXIOS  
    return Object.values(this.arrayNotifications[4]);  
  }  
}
```

```

    } else {
        //Si el tamaño es < 3 Es cuando las notificaciones son obtenidas desde el canal privado,
        es decir mediante Laravel Echo y Pusher
        return Object.values(this.arrayNotificaciones[0]);
    }
}

```

The screenshot shows a web application interface on the left and the Vue.js devtools on the right. The web application has a header with a user profile 'anton' and a notification bell icon. Below the header, there is a 'Notificaciones' section that says 'No tiene notificaciones'. The main content area features a blue bar chart titled 'Ventas' with a label 'Mayo' at the bottom. At the bottom of the page, it says 'Desarrollado por IncanatoIT'.

The Vue.js devtools on the right are open to the 'Vue' tab. It shows the component tree with the following structure:

- <Root>
 - <Notificacion> = \$vm0
 - <Dashboard>

The selected component is <Notificacion>. Its props, data, and computed properties are listed below:

- props
 - notifications: Array[0]
- data
 - arrayNotificaciones: Array[0]
- computed
 - listar: Array[0]