Admin Dashboard Lenses

Admin Dashboard Lenses

What are the default dashboard lenses available?

The default dashboard lenses are stored in the admin configuration file as an array called **dashboard_lenses**. You can add or remove dashboard lens classes from this array.

Key	Class	Permission
revenue-lens	Aero\Admin\Lenses\RevenueLens	reports.orders
average-order-value-lens	Aero\Admin\Lenses\AverageOrderValueLens	reports.orders
top-shipping-countries-lens	Aero\Admin\Lenses\TopShippingCountriesLens	reports.orders
returns-rate-lens	Aero\Admin\Lenses\ReturnsRateLens	reports.orders
orders-lens	Aero\Admin\Lenses\OrdersLens	reports.orders
top-selling-manufacturers-lens	Aero\Admin\Lenses\TopSellingManufacturersLens	reports.catalog
top-selling-items-lens	Aero\Admin\Lenses\TopSellingItemsLens	reports.catalog

What is a dashboard lens?

Dashboard lenses are the blocks of content that make up the interactive part of the admin dashboard.

They provide an overview of various stats such as your total revenue, orders, and your top selling products.

There are a number of default lenses and you can also create your own

How do I create a custom dashboard lens?

To create a dashboard lens you need to create a class that extends `Aero\Admin\AdminLens` and implements the **data** method.

data Method

This method is executed every time the lens needs data (when the dashboard loads or when the dashboard date range is changed). This method must return an **Illuminate\Http\JsonResponse** because it is called on the frontend by javascript.

data Method Parameters

Type	Description
Request	The first parameter is the request.
Array	The second parameter is an array that holds the start and end date selected for the dashboard stats. This array has a start and end key that has the dates.
Array	The third parameter is an array that holds information about the comparison dates. This array has a type key that is 1 for the same period prior, 2 for the same period 1 year ago, or 3 for a custom period. This array has a date key that has the same structure as the second parameter. This array also has a text key that holds text to be displayed to explain the comparison.

Here is the code for our top shipping countries implementation:

```
<?php

namespace Aero\Admin\Lenses;

use Aero\Admin\AdminLens;
use Aero\Cart\Models\Order;
use Illuminate\Http\JsonResponse;
```

```
use Illuminate\Http\Request;
use Illuminate\Support\Str;

class TopShippingCountriesLens extends AdminLens
{
   protected $title = 'Top Shipping Countries';

   protected static $permission = 'reports.orders';

   protected static $containerClass = 'row-span-2';

   protected $view = 'admin::lenses.percentage-list';

   public function data(Request $request, array $date, array $compare): JsonResponse
   {
       $countries = Order::visible()->with('shippingAddress.country')-
>whereHas('shippingAddress')
           ->whereBetween('ordered_at', [$date['start'], $date['end']])->get()
           ->groupBy('shippingAddress.country.name')
           ->map(function ($group, $key) {
               return [
                   'name' => $key,
                   'count' => $group->count(),
                   'percentage' => 0,
               ];
           })->sortByDesc('count')->take(5);

       $total = $countries->reduce(function ($count, $country) {
           return $count + $country['count'];
       }, 0);

       $countries->transform(function ($country) use ($total) {
           $country['value'] = number_format(($country['count'] / $total) * 100, 2);
           $country['text'] = $country['count'].' '.Str::plural('order', $country['count']);
           $country['percentage'] = $country['value'].'%';

           return $country;
       });

       return response()->json([
```

```
            'items' => $countries,
        ]);
    }
}
```

How do I use a custom view for my dashboard lens?

You can optionally make your dashboard lens use a custom view. To do this you need to set the protected `$view` string on your lens to the view you would like to use.

```
<?php

namespace Aero\Admin\Lenses;

use Aero\Admin\AdminLens;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;

class TopShippingCountriesLens extends AdminLens
{
   protected $view = 'admin::lenses.percentage-list';

   public function data(Request $request, array $date, array $compare): JsonResponse
   {
       return response()->json([]);
   }
}
```

Your view will have access to Blade and Vue as it is rendered as a slot. You will have access to the following variables:

Variable	Type	Description
lens.data	Vue	This is an object that holds the response from the data method of your admin lens. You can effectively pass any data you want in the json response and have access to it in Vue through this variable.
lens.date	Vue	This is an object that has start, end and compare keys.

Variable	Type	Description
lens.loading	Vue	This is a boolean that is true while the lens is making an API call to get its data.
lens.notLoading	Vue	This is a boolean that is false unless the lens is making an API call.
lens.noData	Vue	This is a boolean that is true if the store has no orders.
lens.hasData	Vue	This is a boolean that is true if the store has orders.
lens.renderDateGetVariables	Vue	This returns an empty string or the selected date formatted as the $dateGetVariables property on the lens class describes.
$title	Blade	This is the value of the $title property on your lens class.
$link	Blade	This is the value of the link() function on your lens class.
$noTextData	Blade	This is the value of the $noDataText property on your lens class.

Here is the code for our default lens view:

```
<div class="card h-full relative">
   <svg class="w-4 h-4 absolute pin-t pin-r m-4 text-grey-light stroke-current [ animation-
spin ]" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 18.111 18.068" v-if="lens.loading">
       <path d="M20,4V9h-
.582M4.062,11A8,8,0,0,1,19.418,9m0,0H15M4,20V15h.581m0,0a8,8,0,0,0,15.357-2M4.581,15H9"
transform="translate(-2.945 -2.964)" fill="none" stroke-linecap="round" stroke-
linejoin="round" stroke-width="2"/>
   </svg>
   <svg class="w-4 h-4 absolute pin-t pin-r m-4 text-success stroke-current [ animation-fade-
out ]" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 18.11 18.07" v-else>
       <g fill="none" stroke-linecap="round" stroke-linejoin="round" stroke-width="2">
           <path d="M2.06 10.03l4 4 10-10" data-name="Path 103"/>
       </g>
   </svg>

   <h3 class="uppercase text-base font-normal text-text h-auto p-0 mb-3">{{ $title }}</h3>

   <div class="flex flex-wrap -mx-3">
```

```
        <div class="w-1/2 px-3 mb-6" v-if="lens.loading || lens.noData">
            <div class="mb-1">
                <template v-if="lens.loading">
                    <skeleton-box height="2.125rem" width="4rem" />
                </template>
                <template v-else-if="lens.noData">
                    <p class="text-3xl font-medium mb-1">–</p>
                </template>
            </div>
            <template v-if="lens.loading">
                <skeleton-box width="9rem" height="0.8125rem" />
            </template>
            <template v-else-if="lens.noData">
                <p class="text-xs">{{ $noDataText }}</p>
            </template>
        </div>

        <div class="w-1/2 px-3 mb-6" v-if="lens.notLoading && lens.hasData" v-for="data in
lens.data" :key="data.currency_code">
            <div class="mb-1">
                <p class="text-3xl font-medium" v-html="data.text"></p>
            </div>
            <p class="text-xs"><strong v-text="data.compare.percentage" class="font-medium"
:class="{ 'text-success' : data.compare.up, 'text-error' : ! data.compare.up }"></strong>
<span v-text="data.compare.text"></span></p>
        </div>
    </div>

    @isset($link)
        <a class="dashboard-link" v-if="lens.noData">{{ $link['text'] }}</a>
        <a class="dashboard-link" :href="'{{ $link['route'] }}' + lens.renderDateGetVariables"
v-else>{{ $link['text'] }}</a>
    @endif
</div>
```

Adding Permissions to your Dashboard Lens

You can optionally make your dashboard lens require a permission (if the user doesn’t have the permission then the lens will not be rendered). To do this you need to set the protected static $permission string on your lens class to the permission you would like to use.

```
<?php

namespace Aero\Admin\Lenses;

use Aero\Admin\AdminLens;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;

class TopShippingCountriesLens extends AdminLens
{
   protected static $permission = 'dashboard.lens.orders';

   public function data(Request $request, array $date, array $compare): JsonResponse
   {
       return response()->json([]);
   }
}
```

How do I add a custom dashboard lens to the dashboard?

To add your custom dashboard lens to the dashboard you need to register it with the admin and the admin dashboard.

To register your dashboard lens with the admin you need to call the **Aero\Admin\Facades\Admin** facade **registerLens**method and pass in your dashboard lens class.

```
<?php

namespace Acme\MyModule;

use Aero\Admin\Facades\Admin;
use Aero\Admin\Lenses\RevenueLens;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       Admin::registerLens(RevenueLens::class);
   }
}
```

Once you have registered your dashboard lens with the admin you are able to extend **Aero\Admin\Http\Responses\AdminDashboardPage**and add your lens using the **setLens**, **addLens**, **addLensBefore**, or **addLensAfter**methods.

setLens Method

This method accepts a key and a lens class. The key's current value will be replaced with the lens class you provide.

addLens Method

This method accepts a key and a lens class. When using this method your lens will be added as the first dashboard lens.

addLensBefore Method

This method accepts the same parameters as the **addLens**method but additionally accepts a third parameter, the key of the lens this new lens should be added before.

addLensAfter Method

This method accepts the same parameters as the **addLens**method but additionally accepts a third parameter, the key of the lens this new lens should be added after.

Finding the Current Dashboard Lenses Keys

If you do not know the keys of the current dashboard lenses you can use this code snippet to dump the current keys out when visiting the dashboard.

```
<?php

namespace Acme\MyModule;

use Aero\Admin\Http\Responses\AdminDashboardPage;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       AdminDashboardPage::extend(function (AdminDashboardPage $page) {
           dd($page->getLenses()->keys());
       });
```

```
    }
}
```

```
<?php

namespace Acme\MyModule;

use Aero\Admin\Facades\Admin;
use Aero\Admin\Http\Responses\AdminDashboardPage;
use Aero\Admin\Lenses\RevenueLens;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        Admin::registerLens(RevenueLens::class);

        AdminDashboardPage::extend(function (AdminDashboardPage $page) {
            $page->addLensBefore('revenue-lens', RevenueLens::class, 'average-order-value-
lens');
        });
    }
}
```

How do I remove a dashboard lens from the dashboard?

You are able to extend **Aero\Admin\Http\Responses\AdminDashboardPage**and use the **removeLens**method. You will need to pass in the key of the lens that you want to remove.

```
<?php

namespace Acme\MyModule;

use Aero\Admin\Http\Responses\AdminDashboardPage;
use Aero\Admin\Lenses\RevenueLens;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       AdminDashboardPage::extend(function (AdminDashboardPage $page) {
           $page->removeLens('revenue-lens');
       });
   }
}
```