

Admin

- [General](#)
 - [How do I set the page title for the admin?](#)
- [Dashboard](#)
 - [What are the available lenses?](#)
 - [How do I create a custom dashboard lens?](#)
 - [How do I add a custom dashboard lens to the dashboard?](#)
 - [How can I remove an existing lens from the dashboard?](#)
- [Slots](#)
 - [What are the available admin slots?](#)
 - [How do I inject into admin slots?](#)
- [Filters](#)
 - [What filter types are available?](#)
 - [How do I make my filter invertible?](#)
 - [How do I create a custom admin filter?](#)
- [Resource Lists](#)
 - [What are the available resource lists?](#)
 - [How do I create a custom resource list?](#)
 - [How do I add a column to an existing resource list?](#)
 - [How do I add a filter to an existing resource list?](#)
 - [How do I add a sort by to an existing resource list?](#)
 - [How do I create a custom bulk action?](#)
 - [How do I add a bulk action to an existing resource list?](#)
- [Tables](#)

- [What are the available tables?](#)
- [How do I add a column to an existing table?](#)
- [Reports](#)
 - [How do I create a report?](#)
 - [How do I add a column to an existing report?](#)
 - [How do I add a filter to an existing report?](#)
 - [How do I add a lens to an existing report?](#)
- [Mail Notifications](#)
 - [How do I add a custom mail layout?](#)
 - [How do I add attachments to a mailable?](#)
 - [How do I set the priority of a mailable?](#)
 - [How do I override the html content of a mailable?](#)
 - [What is the preview mail action pipeline?](#)
- [Configuration Page](#)
 - [How do I extend the configuration page?](#)
- [Transformers](#)
 - [What are the available transformers?](#)
 - [How do I add custom data to a transformer?](#)
- [Vue](#)
 - [How do I use Vue in the admin?](#)
 - [How do I put Vue in dev mode?](#)
 - [How do I add a custom Vue component to the admin?](#)
- [Product Picker](#)
 - [How do I use the product picker?](#)
- [Permissions](#)
 - [How do I register new permissions?](#)
 - [How do I check for permissions?](#)
- [Tutorials](#)

- [How do I add a custom field to the new and edit product page?](#)

General

How do I set the page title for the admin?

By default the page title for every admin page is generated from the route name. If you would like to manually set the page title you can set a title specifically for your route or you can return an `adminPageTitle` variable to your view (this includes extending a response builder and adding this to the data).

To set the title manually for a specific route you can use the

`Aero\Admin\Http\Middleware\SetTitle::addTitle()` method. This method expects a route name and a page title:

```
\Aero\Admin\Http\Middleware\SetTitle::addTitle('admin.dashboard', 'The Dashboard Title');
```

Dashboard

What are the available lenses?

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
return-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

The default dashboard lenses are stored in the `admin` configuration file, inside the `dashboard_lenses` array.

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 implement the `data` method:

```
class OrdersLens extends \Aero\Admin\AdminLens
{
    protected $title = 'Orders';

    protected static $permission = 'reports.orders';

    protected static $dateGetVariables = [
        'custom' => 'ordered-at-date',
        'start' => 'ordered-at-date-start-date',
        'end' => 'ordered-at-date-end-date',
    ];

    public function data(\Illuminate\Http\Request $request, array $date, array $compare):
    \Illuminate\Http\JsonResponse
    {
        $ordersCount = \Aero\Cart\Models\Order::visible()->whereBetween('ordered_at',
        [$date['start'], $date['end']])->count();

        $ordersCompareCount = \Aero\Cart\Models\Order::visible()->whereBetween('ordered_at',
        [$compare['date']['start'], $compare['date']['end']])->count();

        if ($ordersCompareCount !== 0) {
            $percentage = (($ordersCount - $ordersCompareCount) / ($ordersCompareCount > 0 ?
            $ordersCompareCount : 1)) * 100;
        }

        return response()->json([
            'text' => number_format($ordersCount),
            'compare' => [
```

```

        'text' => $ordersCompareCount === 0 ? 'No data to compare' : $compare['text'],
        'percentage' => isset($percentage) ? ($percentage >= 0 ? '+' : '-') .
number_format($percentage, 2) . '%' : '',
        'up' => isset($percentage) ? $percentage >= 0 : '',
    ],
    ]));
}

protected function link(): ?array
{
    return auth()->user()->can('reports.view') ? [
        'route' => route('admin.report', ['key' => 'order-breakdown']),
        'text' => 'View Report',
    ] : null;
}
}

```

Parameters

Parameter	Type	Description
<code>\$request</code>	<code>\Illuminate\Http\Request</code>	The current request.
<code>\$date</code>	<code>array</code>	Contains the selected <code>start</code> & <code>end</code> dates.
<code>\$compare</code>	<code>array</code>	Contains the compare <code>date</code> (<code>start</code> & <code>end</code> nested instead) and the <code>text</code> .

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

First, you must register your lens:

```
\Aero\Admin\Facades\Admin::registerLens(\Aero\Admin\Lenses\OrdersLens::class);
```

Then you can add it to the Dashboard:

```
\Aero\Admin\Http\Responses\AdminDashboardPage::extend(function
(\Aero\Admin\Http\Responses\AdminDashboardPage $page, \Closure $next) {
    // Add it as the first lens
    $page->addLens(
        \Aero\Admin\Lenses\OrdersLens::key(),
        \Aero\Admin\Lenses\OrdersLens::class
    );

    // Add it after a specific lens
    $page->addLensAfter(
        \Aero\Admin\Lenses\OrdersLens::key(),
        \Aero\Admin\Lenses\OrdersLens::class,
        \Aero\Admin\Lenses\RevenueLens::key()
    );

    // Add it before a specific lens
    $page->addLensBefore(
        \Aero\Admin\Lenses\OrdersLens::key(),
        \Aero\Admin\Lenses\OrdersLens::class,
        \Aero\Admin\Lenses\AverageOrderValueLens::key()
    );
});
```


How can I remove an existing lens from the dashboard?

You can remove an existing lens by using the `removeLens()` method and passing the `key` of the lens you want to remove:

```
\Aero\Admin\Http\Responses\AdminDashboardPage::extend(function
(\Aero\Admin\Http\Responses\AdminDashboardPage $page, \Closure $next) {
    $page->removeLens(
        \Aero\Admin\Lenses\OrdersLens::key()
    );
});
```

Slots

What are the available admin slots?

Layout Slots

- `styles`
- `banners`
- `app`
- `script`

Login Slots

- `login.header`
- `login.form`
- `login.footer`

Sidebar Slots

- `sidebar.menu`
- `sidebar.before`
- `sidebar.after`

Order Management Slots

Orders

- `orders.index.headers.buttons`
- `orders.order.view.header.buttons`

- `orders.order.view.cards.top`
- `orders.order.view.cards.middle`
- `orders.order.view.cards.bottom`
- `orders.order.view.cards.extra.top`
- `orders.order.view.cards.extra.info`
- `orders.order.view.cards.extra.middle`
- `orders.order.view.cards.extra.bottom`
- `orders.order.new.header.buttons`
- `orders.order.new.cards`
- `orders.order.new.extra.sidebar.customer-information`
- `orders.order.new.extra.sidebar.extra-information`
- `orders.order.new.extra.sidebar`
- `orders.order.new.shipping.header.buttons`
- `orders.order.new.shipping.cards`
- `orders.order.new.payment.header.buttons`
- `orders.order.new.payment.cards`
- `orders.order.edit.top`
- `order.returns.new.middle`
- `order.returns.new.cards`
- `orders.items.item.extra.inline`
- `orders.items.item.extra.after`

Fulfillments

- `orders.fulfillments.index.header.buttons`
- `orders.fulfillments.view.header.buttons`
- `orders.fulfillment.new.extra.info`
- `orders.fulfillment.new.cards`
- `orders.fulfillment.new.cards.extra.top`
- `orders.fulfillment.new.cards.extra.middle`
- `orders.fulfillment.new.sidebar`
- `orders.fulfillment.new.cards.extra.bottom`
- `orders.fulfillment.edit.extra.info`
- `orders.fulfillment.edit.cards`
- `orders.fulfillment.edit.cards.extra.top`
- `orders.fulfillment.edit.cards.extra.middle`
- `orders.fulfillment.edit.sidebar`
- `orders.fulfillment.edit.cards.extra.bottom`

Subscriptions

- `subscriptions.index.header.buttons`
- `subscriptions.subscription.view.extra.form`

- `subscriptions.subscription.view.cards`
- `subscriptions.subscription.view.extra.info`
- `subscriptions.subscription.view.extra.sidebar`

Catalog Management Slots

Products

- `catalog.products.index.header.buttons`
- `catalog.products.deleted.headers.buttons`
- `catalog.product.new.header.buttons`
- `catalog.product.new.alerts`
- `catalog.product.new.cards`
- `catalog.product.new.cards.management`
- `catalog.product.new.cards.attribute`
- `catalog.product.new.after-stock-input`
- `catalog.product.new.pricing`
- `catalog.product.new.inventory`
- `catalog.product.new.shipping`
- `catalog.product.new.status-and-visibility`
- `catalog.product.new.variant`
- `catalog.product.new.cards.variant`
- `catalog.product.new.after-images-card`
- `catalog.product.edit.header.buttons`
- `catalog.product.edit.alerts`
- `catalog.product.edit.cards`
- `catalog.product.edit.cards.management`
- `catalog.product.edit.cards.attribute`
- `catalog.product.edit.after-stock-input`
- `catalog.product.edit.pricing`
- `catalog.product.edit.inventory`
- `catalog.product.edit.shipping`
- `catalog.product.edit.status-and-visibility`
- `catalog.product.edit.variant`
- `catalog.product.edit.cards.variant`
- `catalog.product.edit.after-images-card`

Manufacturers

- `catalog.manufacturers.index.header.buttons`
- `catalog.manufacturers.new.header.buttons`

- `catalog.manufacturers.new.cards`
- `catalog.manufacturers.edit.header.buttons`
- `catalog.manufacturers.edit.cards`

Attributes

- `catalog.attribute-groups.index.header.buttons`
- `catalog.attributes.edit.header.buttons`

Categories

- `catalog.categories.index.header.buttons`
- `catalog.categories.new.cards`
- `catalog.categories.edit.header.buttons`
- `catalog.categories.edit.cards`

Tags

- `catalog.tag-groups.index.header.buttons`
- `catalog.tags.new.cards`
- `catalog.tags.edit.header.buttons`
- `catalog.tags.edit.cards`

Collections

- `catalog.collections.index.header.buttons`
- `catalog.collections.new.header.buttons`
- `catalog.collection.new.cards`
- `catalog.collections.edit.header.buttons`
- `catalog.collection.edit.cards`

Customer Management Slots

Customers

- `customers.index.header.buttons`
- `customer.new.cards`

- `customer.edit.cards`

Customer Groups

- `customers.groups.index.header.buttons`
- `customer.groups.new.cards`
- `customer.groups.edit.cards`

Content Management Slots

Combinations

- `content.combinations.index.header.buttons`
- `content.combinations.new.header.buttons`
- `content.combinations.new.cards`
- `content.combinations.edit.header.buttons`
- `content.combinations.edit.cards`
- `combinations-form.listing-page-content`
- `combinations-form.listing-page-options`

Pages

- `content.pages.index.header.buttons`
- `content.page.new.cards`
- `content.pages.edit.header.buttons`
- `content.page.edit.cards`

Blocks

- `content.blocks.index.header.buttons`

Missing Pages

- `content.missing-pages.index.header.buttons`

Redirects

- `content.redirects.index.header.buttons`
- `content.redirects.new.cards`

Discount Management Slots

Discounts

- `discounts.index.header.buttons`
- `discounts.new.cards`
- `discounts.new.extra.sidebar`
- `discounts.edit.cards`
- `discounts.edit.extra.sidebar`

Vouchers

- `discounts.vouchers.index.header.buttons`

Price Lists

- `price-lists.index.header.buttons`

Configuration Slots

Shipping Methods

- `settings.shipping-methods.index.header.buttons`
- `configuration.shipping-methods.new.shipping-rates`
- `configuration.shipping-methods.new.cards`
- `configuration.shipping-methods.edit.shipping-rates`
- `configuration.shipping-methods.edit.cards`

Fulfillment Methods

- `settings.fulfillment-methods.index.header.buttons`
- `configuration.fulfillment-methods.new.cards`

- `configuration.fulfillment-methods.edit.cards`

Payment Methods

- `settings.payment-methods.index.header.buttons`

Order Statuses

- `settings.order-statuses.index.header.buttons`

Order Status Groups

- `settings.order-status-groups.index.header.buttons`

Order Return Reason Groups

- `settings.order-return-reason-groups.index.header.buttons`

Currencies

- `settings.currencies.index.header.buttons`

Countries

- `settings.countries.index.header.buttons`
- `configuration.countries.new.cards`
- `configuration.countries.edit.cards`

Country Groups

- `settings.country-groups.index.header.buttons`
- `configuration.country-groups.new.cards`
- `configuration.country-groups.edit.cards`

Locations

- `settings.locations.index.header.buttons`
- `configuration.locations.new.cards.information`
- `configuration.locations.new.cards`
- `configuration.locations.edit.cards.information`
- `configuration.locations.edit.cards`

Product Types

- `settings.product-types.index.header.buttons`

Mail Notifications

- `settings.mail.index.header.buttons`
- `configuration.mail.new.header.buttons`
- `configuration.mail.edit.header.buttons`

Customer Tax Groups

- `settings.customer-tax-groups.index.header.buttons`

Product Tax Groups

- `settings.product-tax-groups.index.header.buttons`

Tax Rates

- `settings.tax-rates.index.header.buttons`

Tax Rules

- `settings.tax-rules.index.header.buttons`

Setting Definitions

- `setting-definitions.index.header.buttons`

Flags

- `settings.flags.index.header.buttons`

Export Logs

- `settings.export-logs.index.header.buttons`

Users

- `settings.users.index.header.buttons`
- `configuration.users.new.cards`
- `configuration.users.edit.cards`

How do I inject into admin slots?

Admin Slots allow you to add content within existing admin pages. The examples below show how to inject a view into the product edit page.

For a simple use case, simply provide a view:

```
\Aero\Admin\AdminSlot::inject('catalog.product.edit.cards', 'acme::products');
```

For more advanced use cases, instead of passing a view directly, you can provide a callback function:

```
\Aero\Admin\AdminSlot::inject('catalog.product.edit.cards', function ($data) {  
    // add custom variables to the $data array  
  
    return view('acme::products', $data);  
});
```

These injections should typically sit inside the `boot` method of a Service Provider

Filters

What filter types are available?

As of 15/09/2026, there are eleven filter types available:

Type	Description	Invertible?
Dropdown	Allows users to select a single option from a dropdown menu.	Yes
Radio List	Allows users to select exactly one option from a list of radio buttons.	Yes
Checkbox List	Allows users to select one or more options from a list of checkboxes.	Yes
Min Max	Allows users to specify minimum and maximum values for a numeric range.	No
Min Max Price	Allows users to specify minimum and maximum values for a price range.	No
Date Range	Allows users to filter results by selecting a start date and end date.	No
Options Date Range	A Date Range Filter that also allows users to select a predefined date range option, such as Today, Last 7 days, or Last 30 days.	No
Searchable Select	Allows users to search for and select an option from a searchable list.	Yes
Searchable Select When Over Count	Functions as a Checkbox List Filter when there's fewer than a certain number of options, in which case it then functions as a Searchable Select Filter.	Yes
Searchable Select With Modal	A Searchable Select Filter that supports searching via modal.	Yes

Searchable Select With Product Picker	A Searchable Select Filter for products that supports searching via product picker.	Yes
---------------------------------------	---	-----

Dropdown Filter

```
class ProductStockLevelViewAdminFilter extends \Aero\Admin\Filters\DropdownAdminFilter
{
    protected function handleDropdown($selected, $query)
    {
        if ($selected === 'only-out-of-stock') {
            $query->where('stock_level', '<=', 0);
        } elseif ($selected === 'only-in-stock') {
            $query->where('stock_level', '>', 0);
        } elseif ($selected === 'only-negative') {
            $query->where('stock_level', '<', 0);
        }
    }

    protected function dropdowns(): array
    {
        return [
            [
                'name' => 'Show all',
                'value' => '',
            ],
            [
                'name' => 'Only show out of stock',
                'value' => 'only-out-of-stock',
            ],
            [
                'name' => 'Only show in stock',
                'value' => 'only-in-stock',
            ],
            [
                'name' => 'Only show negative stock',
                'value' => 'only-negative',
            ],
        ];
    }
}
```

```

    }

    public function baseName(): string
    {
        return 'View';
    }
}

```

Radio List Filter

```

class OrderFlagStatusAdminFilter extends \Aero\Admin\Filters\RadioListAdminFilter
{
    protected function handleRadioList($selected, $query)
    {
        return $query
            ->when($selected === 'not', function ($query) {
                $query->whereDoesntHave('flags');
            })
            ->when($selected === 'has', function ($query) {
                $query->whereHas('flags');
            });
    }

    protected function radios(): array
    {
        return [
            [
                'value' => 'has',
                'name' => 'Has been flagged',
            ],
            [
                'value' => 'not',
                'name' => 'Has not been flagged',
            ],
        ];
    }

    public function baseName(): string

```

```
{  
    return 'FlagStatus';  
}  
}
```

Checkbox List Filter

```
class RedirectsStatusAdminFilter extends \Aero\Admin\Filters\CheckboxListAdminFilter  
{  
    protected function handleCheckboxList($selected, $query)  
    {  
        $query->whereIn('status', $selected);  
    }  
  
    protected function checkboxes(): array  
    {  
        return [  
            [  
                'id' => '301',  
                'name' => 'Moved Permanently (301)',  
                'url' => $this->getUrlFor('301'),  
            ],  
            [  
                'id' => '302',  
                'name' => 'Found Redirect (302)',  
                'url' => $this->getUrlFor('302'),  
            ],  
            [  
                'id' => '308',  
                'name' => 'Permanent Redirect (308)',  
                'url' => $this->getUrlFor('308'),  
            ],  
        ];  
    }  
  
    protected function baseName(): string  
    {  
        return 'Status';  
    }  
}
```

```
}  
}
```

`getUrlFor` is a helper method provided by the `Aero\Admin\Filters\Traits\AdminFilterHelpers` trait.

Min Max Filter

```
class ProductStockLevelStockAdminFilter extends \Aero\Admin\Filters\MinMaxAdminFilter  
{  
    public function handleMinMax($min, $max, $query)  
    {  
        $query  
            ->when($min !== null, function ($query) use ($min) {  
                $query->where('stock_level', '>=', $min);  
            })  
            ->when($max !== null, function ($query) use ($max) {  
                $query->where('stock_level', '<=', $max);  
            });  
    }  
  
    public function baseName(): string  
    {  
        return 'Stock';  
    }  
}
```

Min Max Price Filter

```
class VoucherBalanceAdminFilter extends \Aero\Admin\Filters\MinMaxPriceAdminFilter  
{  
    public function handleMinMax($min, $max, $query)  
    {  
        return $query  
            ->when($min !== null, function ($query) use ($min) {  
                $query->where('balance', '>=', $min);  
            });  
    }  
}
```

```

    })
    ->when($max !== null, function ($query) use ($max) {
        $query->where('balance', '<=', $max);
    });
}

public function baseName(): string
{
    return 'Balance';
}
}

```

Date Range Filter

```

class CustomerLastLoginAtAdminFilter extends \Aero\Admin\Filters\DateRangeAdminFilter
{
    protected function handleDateRange($startDate, $endDate, $query)
    {
        $query->whereBetween('last_login_at', [$startDate, $endDate]);
    }

    protected function baseName(): string
    {
        return 'LastLoginAtDate';
    }
}

```

Options Date Range Filter

```

class ExportLogCreatedAtDateAdminFilter extends
\Aero\Admin\Filters\OptionsDateRangeAdminFilter
{
    protected $extraDates = true;

    protected function handleDateRange($startDate, $endDate, $query): void
    {

```

```

        $query->whereBetween('created_at', [$startDate, $endDate]);
    }

    protected function baseName(): string
    {
        return 'CreatedAtDate';
    }
}

```

Searchable Select Filter

```

class OrderDiscountAdminFilter extends \Aero\Admin\Filters\SearchableSelectAdminFilter
{
    protected $multiple = true;

    protected function handleSearchableSelect(array $selected, $query): void
    {
        $query->whereHas('discounts', function ($query) use ($selected) {
            $query->whereIn('id', $selected);
        });
    }

    protected function model(): \Aero\Common\Models\Model
    {
        return new \Aero\Cart\Models\Discount();
    }

    protected function modelName(\Aero\Common\Models\Model $model)
    {
        return $model->code ?? ($model->name ?? 'Discount #'.$model->id);
    }

    protected function searchRoute(): string
    {
        return route('admin.discounts.search');
    }

    protected function baseName(): string

```

```
{  
    return 'Discount';  
}  
}
```

Searchable Select When Over Count Filter

```
class CurrencyAdminFilter extends \Aero\Admin\Filters\SearchableSelectWhenOverCountAdminFilter  
{  
    protected function handleSearchableSelectWhenOverCount(array $selected, $query)  
    {  
        $query->whereHas('currency', function ($query) use ($selected) {  
            $query->whereIn('code', $selected);  
        });  
    }  
  
    protected function model(): \Aero\Common\Models\Model  
    {  
        return new \Aero\Common\Models\Currency();  
    }  
  
    protected function modelName(\Aero\Common\Models\Model $model): string  
    {  
        return $model->name . ' (' . $model->code . ')';  
    }  
  
    protected function searchRoute(): string  
    {  
        return route('admin.configuration.currencies.search');  
    }  
  
    protected function whenOverCount(): int  
    {  
        return 10;  
    }  
  
    protected function baseName(): string  
    {
```

```
        return 'Currency';
    }
}
```

Searchable Select With Modal Filter

```
class PriceListCategoryAdminFilter extends
    \Aero\Admin\Filters\SearchableSelectWithModalAdminFilter
{
    protected $multiple = true;

    protected function handleSearchableSelect(array $selected, $query)
    {
        $query->whereHas('entries', function ($query) use ($selected) {
            $query->whereHasMorph('catalog', [\Aero\Catalog\Models\Category::class], function
($query) use ($selected) {
                $query->whereIn('id', $selected);
            });
        });
    }

    protected function model(): \Aero\Common\Models\Model
    {
        return new \Aero\Catalog\Models\Category();
    }

    protected function modelName(\Aero\Common\Models\Model $model)
    {
        return $model->breadcrumb->map->name->implode(' » ');
    }

    protected function searchRoute(): string
    {
        return route('admin.catalog.categories.search');
    }

    protected function modalRoute(): string
    {

```

```
        return route('admin.catalog.categories.search.tree');
    }

    public function baseName(): string
    {
        return 'Category';
    }
}
```

Searchable Select With Product Picker Filter

```
class PriceListProductAdminFilter extends
    \Aero\Admin\Filters\SearchableSelectWithProductPickerAdminFilter
{
    protected function handleSearchableSelect(array $selected, $query)
    {
        $query->whereHas('entries', function ($query) use ($selected) {
            $query->whereHasMorph('catalog', [\Aero\Catalog\Models\Product::class], function
($query) use ($selected) {
                $query->whereIn('id', $selected);
            });
        });
    }

    public function baseName(): string
    {
        return 'Product';
    }
}
```

How do I make my filter invertible?

As of 15/09/2026, the following filters support being inverted:

Type	Handle Method	Inverted Handle Method
Dropdown	handleDropdown	handleInvertedDropdown
Radio List	handleRadioList	handleInvertedRadioList
Checkbox List	handleCheckboxList	handleInvertedCheckboxList
Searchable Select	handleSearchableSelect	handleInvertedSearchableSelect
Searchable Select When Over Count	handleSearchableSelectWhenOverCount	handleInvertedSearchableSelectWhenOverCount
Searchable Select With Modal	handleSearchableSelect	handleInvertedSearchableSelect
Searchable Select With Product Picker	handleSearchableSelect	handleInvertedSearchableSelect

To enable inverted handling, set `$inverted` to `true` on the filter:

```
class RedirectsStatusAdminFilter extends \Aero\Admin\Filters\CheckboxListAdminFilter
{
    protected $invertible = true;

    protected function handleCheckboxList($selected, $query)
    {
        $query->whereIn('status', $selected);
    }

    protected function handleInvertedCheckboxList(array $selected, $query): void
    {
        $query->whereNotIn('status', $selected);
    }

    protected function checkboxes(): array
    {

```

```

return [
    [
        'id' => '301',
        'name' => 'Moved Permanently (301)',
        'url' => $this->getUrlFor('301'),
    ],
    [
        'id' => '302',
        'name' => 'Found Redirect (302)',
        'url' => $this->getUrlFor('302'),
    ],
    [
        'id' => '308',
        'name' => 'Permanent Redirect (308)',
        'url' => $this->getUrlFor('308'),
    ],
];
}

protected function baseName(): string
{
    return 'Status';
}
}

```

The inverted handler is optional for Laravel 10+ (in most cases), as it is automatically generated using `whereNot`.

How do I create a custom admin filter?

To create a custom admin filter you must create a class that extends `\Aero\Admin\Filters\AdminFilter` and implement a `handle` method:

```
class OrderStatusAdminFilter extends \Aero\Admin\Filters\AdminFilter
{
    use \Aero\Admin\Filters\Traits\HasDefault;
    use \Aero\Admin\Filters\Traits\AdminFilterHelpers;

    protected $view = 'admin::filters.order-status';

    protected function selectedOptions(): array
    {
        return request()->input($this->key(), []) ?? [];
    }

    public function handle(\Symfony\Component\HttpFoundation\ParameterBag $parameters, $query)
    {
        $view = $parameters->get('view');

        if ($view === null && strlen($parameters->get('q', ''))) {
            $view = 'all';

            request()->request->set('view', $view);
        } elseif ($view === null) {
            $view = static::getDefault();
        }

        $selectedStatusIds = $view === 'incomplete'
            ? collect()
            : collect($parameters->get($this->key()))
                ->map(function ($value) {
```

```

        return intval($value);
    })
    ->filter()
    ->values();

$query->when($view === 'incomplete', function ($query) {
    $query->whereNull('ordered_at');
})->when($view === 'outstanding' || $view === 'new', function ($query) use
($selectedStatusIds) {
    $outstandingStatusIds =
\Aero\Cart\Models\OrderStatus::forState(...setting('admin.outstanding_order_states'))-
>pluck('id');

    $statusIds = $selectedStatusIds->isNotEmpty()
        ? $outstandingStatusIds->intersect($selectedStatusIds)
        : $outstandingStatusIds;

    if ($statusIds->isNotEmpty()) {
        $query->whereIn('order_status_id', $statusIds);
    }
})->when($view === 'all', function ($query) use ($selectedStatusIds) {
    $query->whereNotNull('ordered_at');

    if ($selectedStatusIds->isNotEmpty()) {
        $query->whereIn('order_status_id', $selectedStatusIds);
    }
});
}

protected function viewData(): array
{
    $selectedStatuses = collect(request()->input($this->key()));
    $selectedView = request()->input('view', static::getDefault());

    $orderStatuses = \Aero\Cart\Models\OrderStatus::ordered()->get()
        ->map(function ($status) use ($selectedStatuses) {
            return [
                'id' => $status->id,
                'name' => $status->name,
                'state' => $status->state,
            ];
        });
}

```

```

        'url' => $this->getUrlFor($status->id),
    ];
});

return [
    'selectedStatuses' => $selectedStatuses,
    'statuses' => collect($orderStatuses)
        ->filter(function ($status) use ($selectedView) {
            switch ($selectedView) {
                case 'all': return $status['state'] !== 'incomplete';
                case 'new':
                case 'outstanding': return in_array($status['state'],
setting('admin.outstanding_order_states')->toArray());
            }

            return false;
        })
        ->toArray(),
];
}

protected function baseName(): string
{
    return 'Status';
}

public function stateFields(): array
{
    return [$this->key(), 'view'];
}

protected static function defaultValue(): string
{
    return 'outstanding';
}
}

```

And then create a view for rendering the filter, in this case `admin::filters.order-status`:

```

<div class="border-b border-grey-light" data-filter-id="status">
    <button class="flex justify-between items-center w-full p-4" type="button"
onClick="toggleContent(this)">
        <span class="text-sm font-medium">{{ $title }}</span>
        <span class="[ js-accordion-arrow ] w-4 h-4 flex items-center justify-center transform
rotate-90">@include('admin::icons.caret-down')</span>
    </button>

    <div class="p-4 pt-2 list-reset [ js-accordion-content ] hidden">
        <div class="select w-full -mt-1">
            <select id="view_all" name="view" class="w-full" autocomplete="off"
onChange="updateUrlQuery('view', this.value)">
                <option value="outstanding" {{ request('view',
\\Aero\\Admin\\Filters\\Order\\OrderStatusAdminFilter::getDefault()) === 'outstanding' ||
request('view', \\Aero\\Admin\\Filters\\Order\\OrderStatusAdminFilter::getDefault()) === 'new' ?
'selected' : '' }} data-class="new">Outstanding Orders</option>
                <option value="all" data-class="all"
                    @if(request('view',
\\Aero\\Admin\\Filters\\Order\\OrderStatusAdminFilter::getDefault()) == 'all') selected data-
filter-active="true" data-filter-group-name="View" data-filter-name="All Orders" data-filter-
key="view" @endif>All Orders</option>
                <option value="incomplete" data-class="incomplete"
                    @if(request('view',
\\Aero\\Admin\\Filters\\Order\\OrderStatusAdminFilter::getDefault()) == 'incomplete') selected
data-filter-active="true" data-filter-group-name="View" data-filter-name="Incomplete Orders"
data-filter-key="view" @endif>Incomplete Orders</option>
            </select>
        </div>

        @if(count($statuses) > 0)
            <div class="pt-4 flex flex-col bg-white mt-2 overflow-hidden">
                <ul class="list-reset overflow-auto leading-loose" style="max-height: 216px">
                    @foreach($statuses as $status)
                        <li class="mb-2 filter">
                            <label for="status-{{ $status['id'] }}"><label class="checkbox">
                                <input id="status-{{ $status['id'] }}" type="checkbox"
name="{{ $key }}" autocomplete="off" value="{{ $status['id'] }}" onclick="location.href =
'{{ $status['url'] }} ?? ' }}" ;"
                                @if($selectedStatuses->contains($status['id']))
checked data-filter-active="true" data-filter-group-name="{{ $title }}" data-filter-name="{{

```

```
$status['name'] }}" data-filter-url="{{ $status['url'] }}" @endif
        > <span></span></label>
        {{ $status['name'] }}
    </label>
</li>
@endforeach
</ul>
</div>
@endif
</div>
</div>
```

Which receives the `viewData` defined by your filter type as well as the `options` defined by the base class (`$title` and `$key`).

Resource Lists

What are the available resource lists?

Class	View
<code>\Aero\Admin\ResourceLists\AttributeGroupsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\BlocksResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\CategoriesResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\CollectionsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\CombinationsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\CountriesResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\CountryGroupsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\CurrenciesResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\CustomerGroupsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\CustomersResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\CustomerTaxGroupsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\DeletedProductsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\DiscountsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\ExportLogsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\FlagsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\FulfillmentMethodsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\FulfillmentsResourceList</code>	<code>admin::resource-lists.index</code>

<code>\Aero\Admin\ResourceLists\LocationsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\MailNotificationsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\ManufacturersResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\MissingPagesResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\OrderReturnReasonGroupsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\OrdersResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\OrderStatusesResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\OrderStatusGroupsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\PagesResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\PaymentMethodsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\PriceListsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\ProductsResourceList</code>	<code>admin::catalog.products.index</code>
<code>\Aero\Admin\ResourceLists\ProductTaxGroupsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\ProductTypesResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\RedirectsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\SettingDefinitionsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\ShippingMethodsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\SubscriptionActionsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\SubscriptionPlansResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\SubscriptionsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\TagGroupsResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\TaxRatesResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\TaxRulesResourceList</code>	<code>admin::resource-lists.index</code>

<code>\Aero\Admin\ResourceLists\UsersResourceList</code>	<code>admin::resource-lists.index</code>
<code>\Aero\Admin\ResourceLists\VouchersResourceList</code>	<code>admin::resource-lists.index</code>

How do I create a custom resource list?

To create a resource list you need a normal Laravel controller and a class that extends

`Aero\Admin\ResourceLists\AbstractResourceList`.

```
class OrdersResourceList extends \Aero\Admin\ResourceLists\AbstractResourceList
{
    //
}
```

Your controller should be like this example but for your new resource list:

```
class OrdersController extends \Aero\Admin\Http\Controllers\Controller
{
    public function index(\Aero\Admin\ResourceLists\OrdersResourceList $list,
        \Illuminate\Http\Request $request)
    {
        return view('admin::resource-lists.index', [
            'list' => $list = $list(),
            'results' => $list->apply($request->all())
                ->with('status', 'customer', 'items', 'currency')
                ->paginate($request->input('per_page', 24) ?? 24),
        ]);
    }
}
```

Columns

To add columns to your resource list you need to implement the protected `columns()` method and make it return an `array` of your columns.

The entries in the array should use the `Aero\Admin\ResourceLists\ResourceListColumn::create()` helper method. This method expects a string parameter for the columns heading, a closure parameter for how to render the columns data (this can return a string), and an optional string parameter for the columns key (by default this is generated from the columns header):

```
class OrdersResourceList extends \Aero\Admin\ResourceLists\AbstractResourceList
{
    protected function columns(): array
    {
        return [
            \Aero\Admin\ResourceLists\ResourceListColumn::create('Order', function ($row) {
                return view('admin::resource-lists.link', [
                    'route' => route('admin.orders.view', array_merge(request()->all(),
                        ['order' => $row])),
                    'text' => '#'.$row->reference,
                ]->render();
            }),
        ];
    }
}
```

Searchable Columns

You can use the `setSearchAction()` method to define how your column is searched:

```
ResourceListColumn::create('Order', function ($row) {
    return view('admin::resource-lists.link', [
        'route' => route('admin.orders.view', array_merge(request()->all(), ['order' =>
            $row])),
        'text' => '#'.$row->reference,
    ]->render();
}))
->setSearchAction(function ($query, $term) {
    $searchTerm = ltrim($search, '#');

    $query->whereLower('reference', 'like', "%{$searchTerm}%");
}, 'Order Reference')
```

Invisible Columns

You can use the `invisible()` method to make a column invisible. This can be useful if wanting to add a search action but not display the relevant data.

Rows

Row views are additional rows rendered after each complete row (the best example of this is how the order items are listed after each row of the order on the orders page). To add a row view you simply need to provide your view in the protected `$rowViews` array:

```
class OrdersResourceList extends \Aero\Admin\ResourceLists\AbstractResourceList
{
    protected $rowViews = [
        'admin::resource-lists.orders.items',
    ];
}
```

Default Search

When your resource list is searched without a specific column being selected a `handleSearch()` method will be called to handle searching.

The method will be passed the search term and you can use `$this->query` to access the query:

```
class OrdersResourceList extends \Aero\Admin\ResourceLists\AbstractResourceList
{
    protected function handleSearch($search)
    {
        $searchTerm = ltrim($search, '#');

        $this->query->whereLower('reference', 'like', "%{$searchTerm}%");
    }
}
```

Buttons

Most resource list pages will require a create button, typically a create resource button. You can easily add buttons to the top of your resource list page by defining a header slot. Once you have defined a header slot you can inject your view using the admin slots helper.

```
class OrdersResourceList extends \Aero\Admin\ResourceLists\AbstractResourceList
{
    protected $headerSlot = 'orders.index.header.buttons';
}
```

See [here](#) for how to inject into admin slots.

Filters

To add a filter to your resource list, simply add it to the `$filters` array:

```
class OrdersResourceList extends \Aero\Admin\ResourceLists\AbstractResourceList
{
    protected $filters = [
        \Aero\Admin\Filters\Order\OrderStatusAdminFilter::class,
    ];
}
```

See [here](#) for how to create an existing admin filter, or [here](#) for how to create a custom filter.

Sort Bys

To add sort bys to your resource list you need to implement the protected `sortBy()` method and make it return an `array` of your sort bys.

The entries in the array should use the `\Aero\Admin\ResourceLists\ResourceListSortBy::create()` helper method. This method expects an array parameter holding the dropdown options that will be shown in the frontend dropdown, and a closure that will be executed when the sort option is active:

```
class OrdersResourceList extends \Aero\Admin\ResourceLists\AbstractResourceList
{
    protected function sortBy(): array
    {
        return [
            // Default sort by
            \Aero\Admin\ResourceLists\ResourceListSortBy::create(null, function ($sortBy,
                $query) {
```

```
        return $query->orderByRaw('coalesce(`orders`.`ordered_at`,
`orders`.`created_at`) desc');
    }),

    \Aero\Admin\ResourceLists\ResourceListSortBy::create([
        'order-az' => 'Order A to Z',
        'order-za' => 'Order Z to A',
    ], function ($sortBy, $query) {
        return $sortBy === 'order-az' ? $query->orderBy('reference') : $query-
>orderByDesc('reference');
    }),
];
}
}
```

How do I add a column to an existing resource list?

You can use the `addColumn()` method(s) to add a column to an existing resource list:

```
\Aero\Admin\ResourceLists\OrdersResourceList::extend(function
(\Aero\Admin\ResourceLists\OrdersResourceList $list, \Closure $next) {
    // Add it as the first column
    $list->addColumn('Header', function($row) {
        return 'Content';
    });

    // Add it after a specific column
    $list->addColumnAfter('Header', function($row) {
        return $row->reference;
    }, 'customer');

    // Add it before a specific column
    $list->addColumnBefore('Header', function($row) {
        return $row->reference;
    }, 'customer');

    return $next($list);
});
```

If you need to see the keys of the existing columns in order to position your new column appropriately, you can dump them out:

```
\Aero\Admin\ResourceLists\OrdersResourceList::extend(function
(\Aero\Admin\ResourceLists\OrdersResourceList $list, \Closure $next) {
    dd($list->getColumns()->map->key());

    return $next($list);
});
```


How do I add a filter to an existing resource list?

You can use the `addFilter()` method to add a filter to an existing resource list:

```
\Aero\Admin\ResourceLists\OrdersResourceList::extend(function
(\Aero\Admin\ResourceLists\OrdersResourceList $list, \Closure $next) {
    $list->addFilter(\Aero\Admin\Filters\Order\OrderStatusAdminFilter::class);

    return $next($list);
});
```

How do I add a sort by to an existing resource list?

You can use the `addSortBy()` method to add a filter to an existing resource list:

```
\Aero\Admin\ResourceLists\OrdersResourceList::extend(function
(\Aero\Admin\ResourceLists\OrdersResourceList $list, \Closure $next) {
    $list->addSortBy(\Aero\Admin\ResourceLists\ResourceListSortBy::create([
        'latest' => 'Latest',
        'oldest' => 'Oldest',
    ]), function ($sortBy, $query) {
        return $sortBy === 'oldest' ? $query->orderBy('created_at') : $query-
>orderByDesc('created_at');
    }));

    return $next($list);
});
```

How do I create a custom bulk action?

You can create a class that extends `\Aero\Admin\BulkActions\BulkActionJob` and implement a `handle` method:

```
class DeleteProductsBulkAction extends \Aero\Admin\BulkActions\BulkActionJob
    implements \Illuminate\Contracts\Queue\ShouldQueue
{
    protected $list;

    public function __construct(\Aero\Admin\ResourceLists\ProductsResourceList $list)
    {
        $this->list = $list;
    }

    public function handle(): void
    {
        $this->list->items()->each(static function (\Aero\Catalog\Models\Product $product) {
            $product->delete();
            event(new \Aero\Catalog\Events\ProductDeleted($product));
        });
    }
}
```

You should implement `\Illuminate\Contracts\Queue\ShouldQueue` if you want the bulk action to be queued.

How do I add a bulk action to an existing resource list?

```
\Aero\Admin\BulkAction::create(  
    \Aero\Admin\BulkActions\DeleteProductsBulkAction::class,  
    \Aero\Admin\ResourceLists\ProductsResourceList::class  
)  
->title('Delete products')  
->permissions('products.delete')  
->confirm();
```

See [here](#) for how to create a custom bulk action.

Methods

Method	Description
<code>notRunnable()</code>	Removes the <code>run</code> button from the bulk action, intention being you replace this with a custom button.
<code>wideContent()</code>	Displays bulk action view in a wider container.
<code>confirm()</code>	Prompts the user to confirm running the bulk action.
<code>confirmTitle(string \$title)</code>	Sets the title of the confirmation popup. Default is "Are you sure?"
<code>confirmText(string \$text)</code>	Sets text in the confirmation popup. Default is no text.
<code>confirmButtonText(string \$text)</code>	Sets the confirmation popup confirm button text. Default is "Yes".
<code>title(string \$title)</code>	Sets the title of the bulk action.
<code>processingText(string \$text)</code>	Sets the processing text (shown when a bulk action is queued). Default is <code>__('Bulk action is being processed')</code> .
<code>successText(string \$text)</code>	Sets the success text (shown when a bulk action is ran). Default is <code>__('Bulk action has been processed')</code> .

<code>view(string \$view)</code>	Sets a view for the bulk action to redirect to when ran. Allows options to be provided that are required for the bulk action to run.
<code>group(string \$group)</code>	Sets the group of the bulk action, used when they're displayed to order them.

Tables

What are the available tables?

Customers

- `Aero\Admin\Tables\Customers\CustomerOrdersTable`
- `Aero\Admin\Tables\Customers\CustomerVouchersTable`

Fulfillments

- `Aero\Admin\Tables\Fulfillments\FulfillmentItemsTable`

Orders

- `Aero\Admin\Tables\Orders\OrderCreateFulfillmentTable`
- `Aero\Admin\Tables\Orders\OrderCreateReturnTable`
- `Aero\Admin\Tables\Orders\OrderFulfilledItemsTable`
- `Aero\Admin\Tables\Orders\OrderReturnedItemsTable`
- `Aero\Admin\Tables\Orders\OrderUnfulfilledItemsTable`

Tags

- `Aero\Admin\Tables\Tags\ManageTagsTable`

How do I add a column to an existing table?

You can use the `add()` method(s) to add a column to an existing table:

```
\Aero\Admin\Tables\Orders\OrderUnfulfilledItemsTable::extend(function
(\Aero\Admin\Tables\Orders\OrderUnfulfilledItemsTable $table) {
    // Add it at the end
    $table->column('reference')
        ->value(function ($row) {
            return $row->reference ?? view('admin::resource-lists.placeholder');
        });

    // Add it after a specific column
    $table->column('reference')
        ->after('sku')
        ->value(function ($row) {
            return $row->reference ?? view('admin::resource-lists.placeholder');
        });

    // Add it before a specific column
    $table->column('reference')
        ->before('sku')
        ->value(function ($row) {
            return $row->reference ?? view('admin::resource-lists.placeholder');
        });
});
```

Reports

How do I create a report?

To create a report, make a class that extends `\Aero\Admin\Reports\Report`.

For example, let's make a basic stock report.

First, let's make the report class, this defines the base query for our report:

```
class StockReport extends \Aero\Admin\Reports\Report
{
    use \Aero\Admin\Reports\Traits\HasTables;
    use \Aero\Admin\Reports\Traits\HasFilters;
    use \Aero\Admin\Reports\Traits\CanExport;

    protected static $exports = [
        \Aero\Admin\Reports\Exporters\CSVExporter::class,
        \Aero\Admin\Reports\Exporters\JSONExporter::class,
        \Aero\Admin\Reports\Exporters\XMLExporter::class,
    ];

    protected static $tables = [
        // TODO
    ];

    protected static $filters = [
        // TODO
    ];

    protected function newQuery(): \Illuminate\Database\Eloquent\Builder
    {
        return \Aero\Catalog\Models\Variant::query()
            ->select([
                'variants.id',
                'variants.sku',
                'variants.stock_level',
            ])
            ->groupBy('variants.id');
```

```
}  
}
```

Next, let's make a table class, this defines the UI of the report:

```
class StockReportTable extends \Aero\Admin\Reports\ReportTable  
{  
    public function columns(): array  
    {  
        $positionStart = 0;  
        $positionIncrement = 10;  
  
        return [  
            \Aero\Admin\Reports\ReportTableColumn::create('SKU', function ($row) {  
                return $row->sku;  
            })  
            ->position($positionStart = $positionStart + $positionIncrement)  
            ->setSearchAction(function ($query, $search) {  
                $query->whereLower('sku', 'like', "%{$search}%");  
            })),  
  
            \Aero\Admin\Reports\ReportTableColumn::create('Stock', function ($row) {  
                return $row->stock_level;  
            })  
            ->position($positionStart = $positionStart + $positionIncrement),  
        ];  
    }  
  
    public function emptyMessage(): string  
    {  
        return 'No stock data';  
    }  
}
```

Next, let's make a stock level filter:

```
class MinMaxStockFilter extends \Aero\Admin\Filters\MinMaxAdminFilter  
{  
    public function handleMinMax($min, $max, $query): void  
    {
```

```

        $query->whereBetween('stock_level', [$min, $max]);
    }

    public function title(): string
    {
        return 'Stock Level';
    }
}

```

Finally, let's hook it all up on the report:

```

class StockReport extends \Aero\Admin\Reports\Report
{
    use \Aero\Admin\Reports\Traits\HasTables;
    use \Aero\Admin\Reports\Traits\HasFilters;
    use \Aero\Admin\Reports\Traits\CanExport;

    protected static $exports = [
        \Aero\Admin\Reports\Exporters\CSVExporter::class,
        \Aero\Admin\Reports\Exporters\JSONExporter::class,
        \Aero\Admin\Reports\Exporters\XMLExporter::class,
    ];

    protected static $tables = [
        \Aerocargo\Acme\Tables\StockReportTable::class,
    ];

    protected static $filters = [
        \Aerocargo\Acme\Filters\MinMaxStockFilter::class,
    ];

    protected function newQuery(): \Illuminate\Database\Eloquent\Builder
    {
        return \Aero\Catalog\Models\Variant::query()
            ->select([
                'variants.id',
                'variants.sku',
                'variants.stock_level',
            ])
            ->groupBy('variants.id');
    }
}

```

```
}  
}
```

And register it so that it shows on the Reports index page:

```
\Aero\Admin\AdminReport::create(\Aerocargo\Acme\StockReport::class)  
  ->title('Stock Report')  
  ->summary('Stock report for all products');
```

How do I add a column to an existing report?

You can use the `addColumn()` method(s) to add a column to an existing report:

```
\Aero\Admin\Reports\Implementations\SalesReport::extend(function
(\Aero\Admin\Reports\Implementations\SalesReport $report, \Closure $next) {
    // Add it as the first column
    $report->table()->addColumn('Header', function ($row) {
        return 'Content';
    });

    // Add it after a specific column
    $report->table()->addColumnAfter('Header', function ($row) {
        return 'Content';
    }, 'name');

    // Add it before a specific column
    $report->table()->addColumnBefore('Header', function ($row) {
        return 'Content';
    }, 'name');

    return $next($report);
});
```

If you need to see the keys of the existing columns in order to position your new column appropriately, you can dump them out:

```
\Aero\Admin\Reports\Implementations\SalesReport::extend(function
(\Aero\Admin\Reports\Implementations\SalesReport $report, \Closure $next) {
    dd($report->table()->getColumns()->map->key());

    return $next($report);
});
```


How do I add a filter to an existing report?

You can use the `addFilter()` method to add a filter to an existing report:

```
\Aero\Admin\Reports\Implementations\SalesReport::extend(function
(\Aero\Admin\Reports\Implementations\SalesReport $report, \Closure $next) {
    $report->addFilter(\Aerocargo\Acme\Filters\MinMaxStockFilter::class);

    return $next($report);
});
```

How do I add a lens to an existing report?

You can use the `addLens()` method to add a lens to an existing report:

```
\Aero\Admin\Reports\Implementations\SalesReport::extend(function
(\Aero\Admin\Reports\Implementations\SalesReport $report, \Closure $next) {
    $report->addLens(\Aerocargo\Acme\Lenses\TotalRevenueLens::class);

    return $next($report);
});
```

```
class TotalRevenueLens extends \Aero\Admin\Reports\ReportLens
{
    protected function value(): \Aero\Cart\CartPrice
    {
        $totalRevenue = \DB::query()
            ->fromSub($this->query, 'results')
            ->selectRaw('SUM(results.revenue + results.tax) as total')
            ->value('total');

        $currency = request()->store()->currency;

        return new \Aero\Cart\CartPrice($currency, $totalRevenue * 100);
    }
}
```

Where the lens added extends `\Aero\Admin\Reports\ReportLens`:

```
class TotalRevenueLens extends \Aero\Admin\Reports\ReportLens
{
    protected function value(): \Aero\Cart\CartPrice
    {
        $totalRevenue = \DB::query()
```

```
        ->fromSub($this->query, 'results')
        ->selectRaw('SUM(results.revenue + results.tax) as total')
        ->value('total');

    $currency = request()->store()->currency;

    return new \Aero\Cart\CartPrice($currency, $totalRevenue * 100);
}
}
```

`$this->query` is a copy of the report query, the `value` method must return a string (or stringable).

Mail Notifications

How do I add a custom mail layout?

To add a custom mail layout you can use the `addLayout()` method on the mail action:

```
\Aero\Events\Models\MailAction::addLayout('acme', 'Acme');
```

Then you can create the `emails.acme` twig view, which will be used to render your layout:

```
Acme {{ body | raw }}
```

How do I add attachments to a mailable?

You can attach files to a mailable by using the `whenSendingAsMail` callback:

```
\Aero\Cart\Events\OrderConfirmation::whenSendingAsMail(function (\Aero\Events\Actions\Mailable
$mailable, \Aero\Cart\Events\OrderConfirmation $event) {
    $pdf = \Aerocargo\PdfInvoices\Invoice::forOrder($event->order, 'invoice');

    $mailable->attachData(
        $pdf->output(),
        "{$event->order->reference}-invoice.pdf",
        [
            'mime' => 'application/pdf',
        ]
    );
});
```

This example uses the `aerocargo/pdf-invoice` module to generate a PDF Invoice for the order.

How do I set the priority of a mailable?

You can set the priority of a Mailable using the `priority()` method.

For example:

```
// For a specific event
\Aero\Cart\Events\OrderConfirmation::whenSendingAsMail(function (\Aero\Events\Actions\Mailable
$mailable, \Aero\Cart\Events\OrderConfirmation $event) {
    $mailable->priority(1);
});

// For all order events
\Aero\Cart\Events\AbstractOrderEvent::whenSendingAsMail(function
(\Aero\Events\Actions\Mailable $mailable, \Aero\Cart\Events\AbstractOrderEvent $event) {
    $mailable->priority(1);
});

// For all events
\Aero\Events\ManagedEvent::whenSendingAsMail(function (\Aero\Events\Actions\Mailable
$mailable, \Aero\Events\ManagedEvent $event) {
    $mailable->priority(1);
});
```

Email priority does not control how quickly it is delivered. It is a sign of importance to the recipient's email client (it may not be honoured by every provider)

How do I override the html content of a mailable?

You can set the HTML of a Mailable using the `html()` method.

For example, the [email builder](#) uses this to render the HTML of the email from the `template`, rather than the `body`:

```
\Aero\Events\ManagedEvent::whenSendingAsMail(function (\Aero\Events\Actions\Mailable
$mailable, \Aero\Events\ManagedEvent $event) {
    $action = $mailable->action();

    if (! empty($action->body)) {
        return;
    }

    $mailable->html(
        \Aero\EmailBuilder\Facades\EmailBuilder::renderTemplate((array) $event, $action-
>getKey())
    );
});
```

What is the preview mail action pipeline?

The Preview Mail Action Pipeline is used to build a temporary `\Aero\Events\Models\MailAction` instance from request data to be previewed.

For example, you can add custom fields to the `\Aero\Events\Models\MailAction` instance:

```
\Aero\Admin\Pipelines\PreviewMailAction::extend(function
(\Aero\Admin\Pipelines\PreviewMailAction $pipeline, \Closure $next) {
    \Aero\EmailBuilder\Facades\EmailBuilder::boot();

    $data = $pipeline->request()->validated();

    $version = \Aero\EmailBuilder\Helpers\PreviewVersion::get($data);

    $template = \Aero\EmailBuilder\Facades\EmailBuilder::getTemplate($data['id'], $version)-
>toArray();

    $pipeline->mailAction()->fill([
        'template_header' => $template['header'] ?? null,
        'template_body' => $template['body'] ?? null,
        'template_footer' => $template['footer'] ?? null,
        'template_settings' => $template['settings'] ?? null,
    ]);

    return $next($pipeline);
});
```

This example is taken from the `aerocommerce/email-builder` module

Parameters

Parameter	Type	Description
<code>\$pipeline</code>	<code>\Aero\Admin\Pipelines\PreviewMailAction</code>	The preview mail action pipeline.
<code>\$next</code>	<code>\Closure</code>	The next step in the pipeline.

Methods

Method	Description
<code>mailAction()</code>	Returns the temporary mail action instance being built.
<code>request()</code>	Returns the request.

Configuration Page

How do I extend the configuration page?

The configuration page can be extended to customise existing configuration groups, add new configurations, or register entirely new configuration groups.

Configuration groups and configurations are built using a pipeline, allowing you to modify the default configuration page without replacing the existing definitions.

The examples below show how to:

- Change the title of an existing configuration group
- Add a new configuration to an existing group
- Edit an existing configuration
- Create and register a new configuration group

Changing the title of a configuration group

```
\Aero\Admin\Configuration\Groups\OrderManagementConfigurationGroup::extend(function
(\Aero\Admin\Configuration\Groups\OrderManagementConfigurationGroup $group, \Closure $next) {
    $group->title('New Title');

    return $next($group)
});
```

Adding a new configuration to a configuration group

```
\Aero\Admin\Configuration\Groups\OrderManagementConfigurationGroup::extend(function
(\Aero\Admin\Configuration\Groups\OrderManagementConfigurationGroup $group, \Closure $next) {
    $group->configuration('acme')
        ->title('Acme')
});
```

```

        ->description('Description goes here...')
        ->link(route('admin.acme'))
        ->showWhen(function () {
            return \Illuminate\Support\Facades\Gate::check('acme.view');
        });

        return $next($group)
    });

```

Editing an existing configuration in a configuration group

```

\Aero\Admin\Configuration\Groups\OrderManagementConfigurationGroup::extend(function
(\Aero\Admin\Configuration\Groups\OrderManagementConfigurationGroup $group, \Closure $next) {
    $group->configuration('order_statuses')
        ->title('New Title')
        ->description('New description goes here...');

    return $next($group)
});

```

Adding a new configuration group to the configuration page

First, create a class that extends the `\Aero\Admin\Configuration\ConfigurationGroup` pipeline and use the `build` method to scaffold the group:

```

class AcmeConfigurationGroup extends \Aero\Admin\Configuration\ConfigurationGroup
{
    protected function build(): \Aero\Admin\Configuration\ConfigurationGroup
    {
        return $this
            ->title('Acme')
            ->configurations([
                Configuration::make('first')
                    ->description('Description for first')
            ])
    }
}

```

```

        ->link(route('admin.acme.first')),

        Configuration::make('second')
        ->description('Description for second')
        ->link(route('admin.acme.second')),

    });

}

}

```

Then add it to the `Aero\Admin\Http\Responses\AdminConfigurationPage`:

```

// Insert it at the end of the configuration page
\Aero\Admin\Http\Responses\AdminConfigurationPage::addGroup(AcmeConfigurationGroup::class);

// Insert it before the order management configuration group
\Aero\Admin\Http\Responses\AdminConfigurationPage::addGroupBefore(
    AcmeConfigurationGroup::class,
    \Aero\Admin\Configuration\Groups\OrderManagementConfigurationGroup::class
);

// Insert it after the order management configuration group
\Aero\Admin\Http\Responses\AdminConfigurationPage::addGroupAfter(
    AcmeConfigurationGroup::class,
    \Aero\Admin\Configuration\Groups\OrderManagementConfigurationGroup::class
);

// Insert it at the index provided
\Aero\Admin\Http\Responses\AdminConfigurationPage::insertGroup(AcmeConfigurationGroup::class,
0);

```

Transformers

What are the available transformers?

Class	Description
<code>\Aero\Admin\Transformers\AttributesTransformer</code>	This transformer is used in the ProductTransformer to transform the product's attribute groups.
<code>\Aero\Admin\Transformers\BaseTagTransformer</code>	This transformer is used to generate the data for a new tag.
<code>\Aero\Admin\Transformers\BaseVariantTransformer</code>	This transformer is used to generate the data for a new variant.
<code>\Aero\Admin\Transformers\DiscountTransformer</code>	This transformer is used to transform a discount for the discount new/edit page.
<code>\Aero\Admin\Transformers\ProductTransformer</code>	This transformer is used to transform a product for the product new/edit page.
<code>\Aero\Admin\Transformers\TagsTransformer</code>	This transformer is used to transform tags for the tag group new/edit page.
<code>\Aero\Admin\Transformers\TagTransformer</code>	This transformer is used to transform a tag in the <code>\Aero\Admin\Transformers\TagsTransformer</code> .
<code>\Aero\Admin\Transformers\VariantTransformer</code>	This transformer is used in the ProductTransformer to transform the product's variants.

How do I add custom data to a transformer?

You can add a custom field using the `add()` method and returning an array:

```
\Aero\Admin\Transformers\ProductTransformer::add(function ($data) {  
    /* @var \Aero\Catalog\Models\Product */  
    $product = $data['product'];  
  
    return [  
        'foo' => 'bar',  
    ];  
});
```

You can add a cast using the `addCast()` method:

```
\Aero\Admin\Transformers\ProductTransformer::addCast('acme', 'integer');
```

Vue

How do I use Vue in the admin?

When in an admin Blade view file you can use any Vue directive (such as `v-for`, `v-if`, `v-on:click` etc).

It is recommended to use the `v-on:click` syntax instead of `@click` syntax to minimise any potential conflicts between Blade & Vue.

Dev Mode

Putting Vue into dev mode allows you to use the Vue dev tools. You can put the Admin Vue into dev mode by putting this code snippet in your Blade view:

```
@push('scripts')
<script>
    window.AeroAdmin.vue.use({
        install(Vue) {
            Vue.config.devtools = true;
        },
    });
</script>
@endpush
```

Extending Vue without a Custom Component

You can extend Vue without using a custom Vue component by using `window.AeroAdmin`.

Adding Data

```
@push('scripts')
<script>
    window.AeroAdmin.addData('myVariable', '{{ old('myVariable', 'value') }}');
    window.AeroAdmin.addData('myOtherVariable', 'Example');
    window.AeroAdmin.addData('myArrayVariable', {!! json_encode(['one', 'two', 'three',
'four']) !!});
</script>
@endpush
```

Adding Computed

```
@push('scripts')
<script>
    window.AeroAdmin.addComputed('computedMethod', function () {
        return ['one', 'two', 'three', 'four'];
    });
</script>
@endpush
```

Adding Methods

```
@push('scripts')
<script>
    window.AeroAdmin.addMethod('test', function (value) {
        console.log(value);
    });

    window.AeroAdmin.addMethod('testing', function (value) {
        this.test(value);
    })
</script>
@endpush
```

Adding Watchers

```
@push('scripts')
<script>
  window.AeroAdmin.addData('foo', 'bar');

  // Normal watcher - fires on change
  window.AeroAdmin.addWatch('foo', function (value) {
    console.log(value);
  });

  // Immediate watcher - fires immediately (and on change)
  window.AeroAdmin.addWatch('foo', {
    handler: function (value) {
      console.log(value);
    },
    immediate: true
  })
</script>
@endpush
```

You can only have one watcher for a given key, if multiple are registered the last one takes priority.

Adding Listeners

```
@push('scripts')
<script>
  window.AeroAdmin.addListener('loaded', function () {
    console.log('Vue has been loaded');
  });
</script>
@endpush
```

How do I put Vue in dev mode?

You can put the Admin Vue into dev mode by putting this code snippet in your Blade view:

```
@push('scripts')
<script>
    window.AeroAdmin.vue.use({
        install(Vue) {
            Vue.config.devtools = true;
        },
    });
</script>
@endpush
```

How do I add a custom Vue component to the admin?

1. You need to add the `assetLinks` function to your modules service provider so that your modules assets are publicly linkable. An example module service provider can be found here:

<https://github.com/aerocargo/admin-example-vue/blob/master/src/ServiceProvider.php>

2. You also need to make your component `.vue` file(s) and your `main.js` file that will register all of the components. An example javascript structure can be found here:

<https://github.com/aerocargo/admin-example-vue/tree/master/resources/js>

3. You need to run `php artisan aero:link` in your terminal to link the assets.
4. Once you've got your files setup, have ran the link command and have built the javascript files (using npm: `npm run dev` or `npm run production`), you need to ensure that your module's components javascript file is loaded and the components are registered in the Blade view where you want to use the component(s).

To do this you need to include the javascript file from your module and then register the components through the `window.AeroAdmin.vue.use` method:

```
@push('scripts')
    <script src="{{ asset(mix('components.js', 'modules/aerocargo/admin-example-vue')) }}"></script>
    <script>
        window.AeroAdmin.vue.use(window.exampleAdminComponents);
    </script>
@endpush
```

After this you'll be able to use your custom Vue component(s) anywhere in the Blade file where you have registered them.

Product Picker

How do I use the product picker?

The Product Picker provides a modal that lets users select a single or many products/variants with quantities.

Parts of the admin use this, e.g. the MOTO Order screen (create order page).

You can use the `<product-picker></product-picker>` Vue component from any admin view inside of your module that extends the official admin layout (`admin::layouts.main`).

When using the Product Picker you need to provide a url for the `get-products-url` prop (this value is usually always the `admin.catalog.products.picker` route) and you need to provide a slot for the button that will be used to open the picker.

For example:

```
@extends('admin::layouts.main')

@section('content')
    <h2>
        <a href="{{ route('admin.modules') }}" class="btn">@include('admin::icons.back')
        Back</a>
        <span class="ml-4">My Module</span>
    </h2>

    <div class="card">
        <product-picker v-slot="picker"
                        get-products-url="{{ route('admin.catalog.products.picker') }}"
                        <a href="#" @click.prevent="picker.open" class="btn">Add Product</a>
        </product-picker>
    </div>
@endsection
```

Props

Prop	Type	Default	Required
resource-name	string	products	No
singular-resource-name	string	product	No
get-products-url	string	-	Yes
initial-parameters	object	{}	No
initial-selection	object	{}	No
image-url-prefix	string	window.ImageFactory['60']	No
no-image-url	string	window.ImageFactory['no-image']	No
action-name	string	Add Products	No
clear-selected-after-emit	bool	true	No
params	object	{}	No
pick-product	bool	false	No
product-only	bool	false	No
pick-only	bool	false	No
needs-quantity	bool	true	No
results-accessor	string	results	No
emit-image	bool	false	No
emit-sku	bool	false	No
emit-model	bool	false	No
emit-name	bool	false	No
emit-price	bool	false	No
emit-raw-price	bool	false	No
emit-currency	bool	false	No
emit-all	bool	false	No
max-selection	?int	null	No
show-top-slot	bool	true	No
default-search-term	?string	null	No
show-customer-price	bool	false	No
show-cost-price	bool	false	No

resource-name

This is used in various places in the product picker UI, e.g.: `Loading <span v-text="resourceName"></span>`. This is set by default to "products".

singular-resource-name

This is used in various places in the product picker UI, e.g.: `:title="picker.isSelected('p:' + product.id + '-v:') ? ('Remove ' + singularResourceName.toLowerCase()) : ('Add ' + singularResourceName.toLowerCase())"`. This is set by default to "product".

get-products-url

This is the only strictly required parameter, it is the URL that the products data will be fetched from. It's common to set this value to `{{ route('admin.catalog.products.picker') }}`.

initial-parameters

The initial parameters, e.g. `{ active_status: 1 }` will apply the active filter by default. Defaults to `{}`.

initial-selection

The initial selection of products. Defaults to `{}`.

image-url-prefix

This is the image factory prefix to use for the product image URLs. This is set by default to use `{{ image_factory(60, 60)->contain() }}`.

no-image-url

This is the image URL to use when a product doesn't have an image. This is set by default to use `{{ asset('modules/aerocommerce/admin/no-image.svg') }}`.

action-name

This is the button text shown on the confirm button of the product picker modal (bottom right). This is set by default to "Add Products".

clear-selected-after-emit

This clears selected products after the data has been emitted (happens after the selection has been confirmed). This is set to `true` by default.

params

This is sent alongside the fetch request for products as `params`, e.g.:

```
{{ json_encode([
    'country_code' => $countryCode,
    'zone_id' => $zoneId,
    'postcode' => $postcode,
]) }}
```

pick-product

When `true` products will be allowed to be selected as well as variants. This is set to `false` by default.

product-only

When `true` variants will not be listed and not selectable. This is set to `false` by default.

pick-only

When `true` variants will not need to be in stock to be selectable. This is set to `false` by default.

needs-quantity

When `false` a quantity will not be able to be set for the selected variants/products. This is set to `true` by default.

results-accessor

The property that the data resides (returned from the `get-products-url`). Defaults to "results".

emit-image

When ☐ true the products/variants image will be emitted when selected. This is set to ☐ false by default.

emit-sku

When ☐ true the variants sku will be emitted when selected. This is set to ☐ false by default.

emit-modal

When ☐ true the products/variants model will be emitted when selected. This is set to ☐ false by default.

emit-name

When ☐ true the products/variants name will be emitted when selected. This is set to ☐ false by default.

emit-price

When ☐ true the products/variants price will be emitted when selected. This is set to ☐ false by default.

emit-raw-price

When ☐ true the products/variants raw price will be emitted when selected. This is set to ☐ false by default.

emit-currency

When ☐ true the currency of the products/variants price will be emitted when selected. This is set to ☐ false by default.

emit-all

When ☐ true all products/variants data will be emitted when selected. This is set to ☐ false by default.

max-selection

The maximum number of products that can be selected. This is set to `null` by default, meaning there is no limit.

show-top-slot

When `false` the header of the products picker modal is hidden. This is set to `true` by default.

default-search-term

The default search term applied. This is set to `null` by default, meaning none.

show-customer-price

When `true` shows the prices as the customer (resolved from the current cart instance) would see them. This is set to `false` by default.

show-cost-price

When `true` shows a "Cost Price" column next to the "Stock" column. This is set to `false` by default.

Events

Event	Description
<code>products-selected</code>	This event emits when the action button is clicked (bottom right of the modal) and provides an array of all of the selected products/variants.
<code>selected</code>	This event emits when the action button is clicked (bottom right of the modal) and is emitted for each product/variant that has been selected - only a single product/variant is provided.

Permissions

How do I register new permissions?

You can add new permissions by using `\Aero\Admin\Permissions`:

```
// Add a single permission
\Aero\Admin\Permissions::add('acme.view');

// Add multiple permissions
\Aero\Admin\Permissions::add([
    'acme.view',
    'acme.add',
    'acme.edit',
    'acme.delete',
]);
```

How do I check for permissions?

Blade

```
@can('acme.view')  
    ...  
@endcan  
  
@canany(['acme.add', 'acme.edit'])  
    ...  
@endcan
```

Routes

```
Route::get('/acme', [AcmeController::class, 'index'])  
    ->name('admin.modules.acme')  
    ->middleware('can:acme.view');
```

Controllers

```
if ($this->can('acme.view')) {  
    // ...  
}  
  
if ($this->canAny(['acme.add', 'acme.edit'])) {  
    // ...  
}
```

Tutorials

How do I add a custom field to the new and edit product page?

In this mini tutorial we will add a notes field to the product new and edit page for the product and each variant.

This mini tutorial assumes that you have a module setup or you're happy working from the app service provider (using the `boot` method). You can see how to set up a module [here](#).

1. Adding the database migration

To get started we will add a migration file that will update the products and variants tables to have a notes column:

```
class AddNotesFieldsToProductsAndVariants extends \Illuminate\Database\Migrations\Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        \Illuminate\Support\Facades\Schema::table('products', function
        (\Illuminate\Database\Schema\Blueprint $table) {
            $table->string('notes')->nullable()->after('description');
        });

        \Illuminate\Support\Facades\Schema::table('variants', function
```

```
(\Illuminate\Database\Schema\Blueprint $table) {
    $table->string('notes')->nullable()->after('name');
});

}

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    \Illuminate\Support\Facades\Schema::table('products', function
(\Illuminate\Database\Schema\Blueprint $table) {
        $table->dropColumn('notes');
    });

    \Illuminate\Support\Facades\Schema::table('variants', function
(\Illuminate\Database\Schema\Blueprint $table) {
        $table->dropColumn('notes');
    });
}
}
```

2. Adding the slot view

Now we'll create two views, one that will be injected for the product and one that will be injected for the variants. These views will have a text area in them for the notes input.

product-notes-field.blade.php

```
<div class="card mt-4">
    <label for="notes" class="block mb-2">Notes</label>
    <textarea name="notes" id="notes" v-model="product.notes" class="w-full"></textarea>
</div>
```

variant-notes-field.blade.php

```
<div class="p-4" v-if="!isSimpleProduct">
  <label :for="'variant-notes-' + key" class="block mb-2">Notes</label>
  <textarea :name="'variants[' + key + '][notes]'" :id="'variant-notes-' + key" v-
model="variants[key].notes" class="w-full"></textarea>
</div>
```

After creating the views we will inject them into the relevant slots using the

`Aero\Admin\AdminSlot::inject()` method:

```
\Aero\Admin\AdminSlot::inject('catalog.product.new.cards', 'acme::product-notes-field');
\Aero\Admin\AdminSlot::inject('catalog.product.edit.cards', 'acme::product-notes-field');

\Aero\Admin\AdminSlot::inject('catalog.product.new.variant', 'acme::variant-notes-field');
\Aero\Admin\AdminSlot::inject('catalog.product.edit.variant', 'acme::variant-notes-field');
```

3. Making the notes save

3.1. Make the notes fillable

We need to make the notes fillable as Laravel will only mass assign [fillable attributes](#):

```
\Aero\Catalog\Models\Product::makeFillable('notes');
\Aero\Catalog\Models\Variant::makeFillable('notes');
```

3.2. Adding validation for notes field

To validate the notes, we need to extend the product create and update requests:

```
\Aero\Admin\Http\Requests\Catalog\CreateProductRequest::expects('notes', 'nullable|string');
\Aero\Admin\Http\Requests\Catalog\UpdateProductRequest::expects('notes', 'nullable|string');

\Aero\Admin\Http\Requests\Catalog\CreateProductRequest::expects('variants.*.notes',
'nullable|string');
\Aero\Admin\Http\Requests\Catalog\UpdateProductRequest::expects('variants.*.notes',
'nullable|string');
```

See [here](#) for more guidance on extending requests.

3.3. Adding notes to the transformers

Adding notes to the transformers will ensure that it's available in our views through Vue:

```
\Aero\Admin\Transformers\ProductTransformer::add(function ($data) {
    return [
        'notes' => $data['product']->notes ?? '',
    ];
});

\Aero\Admin\Transformers\VariantTransformer::add(function ($data) {
    return [
        'notes' => $data['variant']->notes ?? '',
    ];
});

\Aero\Admin\Transformers\BaseVariantTransformer::add(function ($data) {
    return [
        'notes' => '',
    ];
});
```

See [here](#) for more information about transformers.

Complete Service Provider

```
<?php

namespace Aerocargo\Acme;

use Aero\Admin\AdminSlot;
use Aero\Admin\Http\Requests\Catalog\CreateProductRequest;
use Aero\Admin\Http\Requests\Catalog\UpdateProductRequest;
use Aero\Admin\Transformers\BaseVariantTransformer;
```

```
use Aero\Admin\Transformers\ProductTransformer;
use Aero\Admin\Transformers\VariantTransformer;
use Aero\Catalog\Models\Product;
use Aero\Catalog\Models\Variant;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        if ($this->app->runningInConsole()) {
            $this->loadMigrationsFrom(__DIR__.'/../database/migrations');
        }

        $this->loadViewsFrom(__DIR__.'/../resources/views', 'acme');

        AdminSlot::inject('catalog.product.new.cards', 'acme::product-notes-field');
        AdminSlot::inject('catalog.product.edit.cards', 'acme::product-notes-field');

        AdminSlot::inject('catalog.product.new.variant', 'acme::variant-notes-field');
        AdminSlot::inject('catalog.product.edit.variant', 'acme::variant-notes-field');

        Product::makeFillable('notes');
        Variant::makeFillable('notes');

        CreateProductRequest::expects('notes', 'nullable|string');
        UpdateProductRequest::expects('notes', 'nullable|string');

        CreateProductRequest::expects('variants.*.notes', 'nullable|string');
        UpdateProductRequest::expects('variants.*.notes', 'nullable|string');

        ProductTransformer::add(function ($data) {
            return [
                'notes' => $data['product']->notes ?? '',
            ];
        });

        VariantTransformer::add(function ($data) {
            return [
                'notes' => $data['variant']->notes ?? '',
            ];
        });
    }
}
```

```
});
```

```
BaseVariantTransformer::add(function ($data) {
```

```
    return [
```

```
        'notes' => '',
```

```
    ];
```

```
});
```

```
}
```

```
}
```