

General

- [Requests](#)
 - [What are the available requests?](#)
 - [How can I extend an existing request?](#)
- [Pipelines](#)
 - [What are the available pipelines?](#)
 - [How do I extend pipelines?](#)
- [Response Builders](#)
 - [What are the available response builders?](#)
 - [How do I pass data to a response?](#)
 - [How do I redirect a response?](#)
 - [How do I set the response status code?](#)
 - [How do I set response headers?](#)
 - [How do I attach middleware to a response builder?](#)
 - [How do I change the response view?](#)
- [Services](#)
 - [What is the tax accountant service?](#)
 - [What is the combination serializer service?](#)
 - [What is the login blocker service?](#)
 - [What is the volume service?](#)
 - [What is the weight service](#)
- [Commands](#)
 - [What are the available commands?](#)

- [Tutorials](#)

- [How do I add a custom price to a product with a module?](#)

Requests

What are the available requests?

Storefront

Listed below are all the storefront validation requests:

- Aero\Store\Http\Requests\SearchRequest
- Aero\Store\Http\Requests\CartDiscountCodeApply
- Aero\Store\Http\Requests\CreateAccountRequest
- Aero\Store\Http\Requests\UpdateAccountDetailsRequest
- Aero\Store\Http\Requests\UpdateAccountPasswordRequest
- Aero\Store\Http\Requests\CreateAddressRequest
- Aero\Store\Http\Requests\UpdateAddressRequest

Admin

Listed below are all the admin validation requests:

General

- Aero\Admin\Http\Requests\CombinationFormRequest
- Aero\Admin\Http\Requests\SeoFormRequest
- Aero\Admin\Http\Requests\FileUploadRequest
- Aero\Admin\Http\Requests\ImageUploadRequest
- Aero\Admin\Http\Requests\CreateCommentRequest
- Aero\Admin\Http\Requests\FilterSaveRequest

Catalog Management

- Aero\Admin\Http\Requests\Catalog\CreateAttributeGroupRequest

- Aero\Admin\Http\Requests\Catalog\UpdateAttributeGroupRequest
- Aero\Admin\Http\Requests\Catalog\CreateCategoryRequest
- Aero\Admin\Http\Requests\Catalog\UpdateCategoryRequest
- Aero\Admin\Http\Requests\Catalog\CreateCollectionRequest
- Aero\Admin\Http\Requests\Catalog\UpdateCollectionRequest
- Aero\Admin\Http\Requests\Catalog\CreateManufacturerRequest
- Aero\Admin\Http\Requests\Catalog\UpdateManufacturerRequest
- Aero\Admin\Http\Requests\Catalog\CreateProductRequest
- Aero\Admin\Http\Requests\Catalog\UpdateProductRequest
- Aero\Admin\Http\Requests\Catalog\CreateTagGroupRequest
- Aero\Admin\Http\Requests\Catalog\UpdateTagGroupRequest

Content Management

- Aero\Admin\Http\Requests\Combinations\StoreRequest
- Aero\Admin\Http\Requests\Combinations\UpdateRequest
- Aero\Admin\Http\Requests\Content\Redirects\Store
- Aero\Admin\Http\Requests\Content\Redirects\Update
- Aero\Admin\Http\Requests\Content\CreateBlockRequest
- Aero\Admin\Http\Requests\Content\UpdateBlockRequest
- Aero\Admin\Http\Requests\Content\CreatePageRequest
- Aero\Admin\Http\Requests\Content\UpdatePageRequest

Customer Management

- Aero\Admin\Http\Requests\Customers\CreateRequest
- Aero\Admin\Http\Requests\Customers\UpdateRequest
- Aero\Admin\Http\Requests\Customers\CreateCustomerGroupRequest
- Aero\Admin\Http\Requests\Customers\UpdateCustomerGroupRequest
- Aero\Admin\Http\Requests\Customers\UpdateAddressRequest
- Aero\Admin\Http\Requests\Customers\LockCustomerRequest

Discount Management

- Aero\Admin\Http\Requests\Discounts\CreateDiscountRequest
- Aero\Admin\Http\Requests\Discounts\UpdateDiscountRequest
- Aero\Admin\Http\Requests\Discounts\CreateVoucherRequest
- Aero\Admin\Http\Requests\Discounts\UpdateVoucherRequest
- Aero\Admin\Http\Requests\Discounts\CreatePriceListRequest

Order Management

- Aero\Admin\Http\Requests\Orders\CreateOrderRequest
- Aero\Admin\Http\Requests\Orders>EditOrderRequest
- Aero\Admin\Http\Requests\Orders\CreateFulfillmentRequest
- Aero\Admin\Http\Requests\Orders\UpdateFulfillmentRequest
- Aero\Admin\Http\Requests\Orders\CreateReturnRequest
- Aero\Admin\Http\Requests\Orders\PaymentCaptureRequest
- Aero\Admin\Http\Requests\Orders\PaymentRefundRequest
- Aero\Admin\Http\Requests\Orders\PaymentReleaseRequest
- Aero\Admin\Http\Requests\Orders\ProcessFulfillmentsRequest
- Aero\Admin\Http\Requests\Orders\UpdateBillingAddressRequest
- Aero\Admin\Http\Requests\Orders\UpdateCustomerRequest
- Aero\Admin\Http\Requests\Orders\UpdateDiscountRequest
- Aero\Admin\Http\Requests\Orders\UpdateFlagsRequest
- Aero\Admin\Http\Requests\Orders\UpdateFulfillmentRequest
- Aero\Admin\Http\Requests\Orders\UpdateShippingAddressRequest
- Aero\Admin\Http\Requests\Orders\UpdateShippingMethodRequest
- Aero\Admin\Http\Requests\Orders\UpdateSubscriptionRequest
- Aero\Admin\Http\Requests\Orders\ChangeOrderStatusRequest

Reports

- Aero\Admin\Http\Requests\Reports\ExportReportRequest

Two Factor Authentication

- Aero\Admin\Http\Requests\TwoFactorAuthentication\TwoFactorAuthenticationRequest

Configuration

- Aero\Admin\Http\Requests\Settings\CreateAdminRequest
- Aero\Admin\Http\Requests\Settings\UpdateAdminRequest
- Aero\Admin\Http\Requests\Settings\CreateCountryGroupRequest
- Aero\Admin\Http\Requests\Settings\UpdateCountryGroupRequest
- Aero\Admin\Http\Requests\Settings\CreateCountryRequest
- Aero\Admin\Http\Requests\Settings\UpdateCountryRequest
- Aero\Admin\Http\Requests\Settings\CreateCurrencyRequest
- Aero\Admin\Http\Requests\Settings\UpdateCurrencyRequest
- Aero\Admin\Http\Requests\Settings\CreateCustomerTaxGroupRequest
- Aero\Admin\Http\Requests\Settings\UpdateCustomerTaxGroupRequest
- Aero\Admin\Http\Requests\Settings\CreateFlagRequest
- Aero\Admin\Http\Requests\Settings\UpdateFlagRequest

- Aero\Admin\Http\Requests\Settings\CreateFulfillmentMethodRequest
- Aero\Admin\Http\Requests\Settings\UpdateFulfillmentMethodRequest
- Aero\Admin\Http\Requests\Settings\CreateLocationRequest
- Aero\Admin\Http\Requests\Settings\UpdateLocationRequest
- Aero\Admin\Http\Requests\Settings\CreateMailRequest
- Aero\Admin\Http\Requests\Settings\UpdateMailRequest
- Aero\Admin\Http\Requests\Settings\CreateOrderReturnReasonGroupRequest
- Aero\Admin\Http\Requests\Settings\UpdateOrderReturnReasonGroupRequest
- Aero\Admin\Http\Requests\Settings\CreateOrderStatusGroupRequest
- Aero\Admin\Http\Requests\Settings\UpdateOrderStatusGroupRequest
- Aero\Admin\Http\Requests\Settings\CreatePaymentMethodRequest
- Aero\Admin\Http\Requests\Settings\UpdatePaymentMethodRequest
- Aero\Admin\Http\Requests\Settings\CreateProductTaxGroupRequest
- Aero\Admin\Http\Requests\Settings\UpdateProductTaxGroupRequest
- Aero\Admin\Http\Requests\Settings\CreateSettingDefinitionRequest
- Aero\Admin\Http\Requests\Settings\UpdateSettingDefinitionRequest
- Aero\Admin\Http\Requests\Settings\CreateShippingMethodRequest
- Aero\Admin\Http\Requests\Settings\UpdateShippingMethodRequest
- Aero\Admin\Http\Requests\Settings\CreateSubscriptionPlanRequest
- Aero\Admin\Http\Requests\Settings\UpdateSubscriptionPlanRequest
- Aero\Admin\Http\Requests\Settings\CreateTaxRateRequest
- Aero\Admin\Http\Requests\Settings\UpdateTaxRateRequest
- Aero\Admin\Http\Requests\Settings\CreateTaxRuleRequest
- Aero\Admin\Http\Requests\Settings\UpdateTaxRuleRequest
- Aero\Admin\Http\Requests\Settings\ManageSearchIndexRequest
- Aero\Admin\Http\Requests\Settings\PreviewMailRequest
- Aero\Admin\Http\Requests\Settings\StoreProductTypeRequest

Checkout

Listed below are all the checkout validation requests:

- Aero\Checkout\Http\Requests\ValidateCartCustomer
- Aero\Checkout\Http\Request\ValidateGuestConversionRequest
- Aero\Checkout\Http\Request\ValidateOrderAddress
- Aero\Checkout\Http\Request\ValidatePaymentMethod
- Aero\Checkout\Http\Request\ValidateShippingMethod

Account Area

Listed below are all the account area validation requests:

- Aero\AccountArea\Http\Requests\ValidateAccountDetails
- Aero\AccountArea\Http\Requests\ValidateAddress
- Aero\AccountArea\Http\Requests\ValidateEmail
- Aero\AccountArea\Http\Requests\ValidateLogin
- Aero\AccountArea\Http\Requests\ValidatePasswordReset
- Aero\AccountArea\Http\Requests\ValidatePasswordUpdate
- Aero\AccountArea\Http\Requests\ValidateRegister

How can I extend an existing request?

The `expects()` method can be used to add custom validation rules (and messages) for your custom field (or to overwrite the existing rules/messages for an existing field):

```
// Adding a custom field
\Aero\Admin\Http\Requests\Catalog\CreateProductRequest::expects('acme',
    'required|string|max:255', [
        'acme.required' => 'The acme field is required.',
    ]);

// Making existing categories field required
\Aero\Admin\Http\Requests\Catalog\CreateProductRequest::expects('categories',
    'required|array');
```

Parameters

Parameter	Type	Description
<code>\$field</code>	<code>string</code>	The key of the field to validate.
<code>\$rules</code>	<code>string array \Closure</code>	The rules to use when validating. If callback is supplied, it will receive the request as the argument.
<code>\$messages</code>	<code>array null</code>	The messages for the rules, key's are of the form <code>{ \$field }. { \$rule }</code> .

Pipelines

What are the available pipelines?

Pipeline	Description
<code>\Aero\Store\Pipelines\ContentForBody</code>	Builds content string to be injected into the body of all pages.
<code>\Aero\Store\Pipelines\ContentForHead</code>	Builds content string to be injected into the head of all pages.
<code>\Aero\Store\Pipelines\RobotsTxt</code>	Used to defined/generate a robots.txt file.
<code>\Aero\Store\Pipelines\CartItemBuilder</code>	Run whenever a cart item is added to the cart in order to build the item from the request data.
<code>\Aero\Cart\Pipelines\OrderItemBuilder</code>	Run whenever a cart item is transformed into an order item.
<code>\Aero\Subscription\Pipelines\SubscriptionOrderBuilder</code>	Run when a subscription is processed and requires a new order to be created.
<code>\Aero\Admin\Pipelines\ListingRulesContextBuilder</code>	Builds listing rule context for each rule type.
<code>\Aero\Admin\Pipelines\ListingRulesDataFormatter</code>	Formats listing rules for admin UI.
<code>\Aero\Admin\Pipelines\PreviewMailAction</code>	Builds a temporary mail action to be previewed.
<code>\Aero\Admin\Reports\Pipelines\BuyXDiscountTierReportQuery</code>	Builds the query for the buy x discount tier report.
<code>\Aero\Admin\Reports\Pipelines\DiscountCodesReportQuery</code>	Builds the query for the discount codes report.
<code>\Aero\Admin\Reports\Pipelines\OrderBreakdownReportQuery</code>	Builds the query for the order breakdown report.
<code>\Aero\Admin\Reports\Pipelines\OrderSurchargeReportQuery</code>	Builds the query for the order surcharge report.
<code>\Aero\Admin\Reports\Pipelines\ProductStockLevelReportQuery</code>	Builds the query for the product stock level report.
<code>\Aero\Admin\Reports\Pipelines\RefundReportQuery</code>	Builds the query for the refund report.

<code>\Aero\Admin\Reports\Pipelines\SalesReportQuery</code>	Builds the query for the sales report.
<code>\Aero\Admin\Reports\Pipelines\VoucherReportQuery</code>	Builds the query for the voucher report.
<code>Aero\Admin\Configuration\Groups\PaymentAndShippingConfigurationGroup</code>	Builds the payment and shipping configuration group.
<code>Aero\Admin\Configuration\Groups\OrderManagementConfigurationGroup</code>	Builds the order management configuration group.
<code>Aero\Admin\Configuration\Groups\LocalizationConfigurationGroup</code>	Builds the localization configuration group.
<code>Aero\Admin\Configuration\Groups\ProductManagementConfigurationGroup</code>	Builds the product management configuration group.
<code>Aero\Admin\Configuration\Groups\NotificationsConfigurationGroup</code>	Builds the notifications configuration group.
<code>Aero\Admin\Configuration\Groups\TaxConfigurationGroup</code>	Builds the tax configuration group.
<code>Aero\Admin\Configuration\Groups\SystemConfigurationGroup</code>	Builds the system configuration group.
<code>Aero\Admin\Configuration\Groups\UserManagementConfigurationGroup</code>	Builds the user management configuration group.
<code>Aero\Admin\Configuration\Groups\MaintenanceConfigurationGroup</code>	Builds the maintenance configuration group.
<code>Aero\DataPort\Commands\Pipelines\ImportBlockCSVPipeline</code>	Creates a block from CSV data.
<code>Aero\DataPort\Commands\Pipelines\ImportProductCSVPipeline</code>	Creates a product from CSV data.
<code>\Aero\Store\Pipelines\ListingsPageQuery</code>	Builds the query for the listings page.
<code>Aero\Search\Elastic\Pipelines\ProductAttributes</code>	Builds the attributes for the ES Product model
<code>Aero\Search\Elastic\Pipelines\ListingAttributes</code>	Builds the attributes for the ES Listing model
<code>Aero\Search\Elastic\Pipelines\ListingCategory</code>	Builds listing documents <code>categories</code> data.
<code>Aero\Search\Elastic\Pipelines\ListingCategoryFacet</code>	Builds listing documents category string facet data.
<code>Aero\Search\Elastic\Pipelines\ListingManufacturerFacet</code>	Builds listing documents manufacturer string facet data.

<code>Aero\Search\Elastic\Pipelines\ListingTagFacet</code>	Builds listing documents tag string facet data.
<code>Aero\Search\Elastic\Pipelines\Mapping</code>	Used to build the ES Mapping for search documents.

How do I extend pipelines?

The payload data can be altered using the `extend()` method. This is typically done from within a Service Provider:

- If the code is unique to a particular store it could be placed in the `boot()` method of the `AppServiceProvider`.
- If it is to be included as part of a module, then it can be added to the setup method of the module's Service Provider.

For example, if a module needs to add an option to items to flag them as requiring gift wrap, the `CartItemBuilder` pipeline class should be extended in the module's ServiceProvider as shown below:

```
<?php

namespace Aerocargo\Acme;

use Aero\Cart\CartItemOption;
use Aero\Common\Providers\ModuleServiceProvider;
use Aero\Store\Pipelines\CartItemBuilder;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        CartItemBuilder::extend(function ($item) {
            if (request()->has('gift_wrap')) {
                $option = CartItemOption::create('Gift wrap');

                $item->addOption($option);
            }
        });
    }
}
```


Response Builders

What are the available response builders?

Storefront

Listed below are all the storefront response builders, the vast majority are view page responses or cart action responses:

- Aero\Store\Http\Responses\AccountOverviewPage
- Aero\Store\Http\Response\CartDiscountCodeApply
- Aero\Store\Http\Response\CartDiscountCodeRemove
- Aero\Store\Http\Response\CartEmpty
- Aero\Store\Http\Response\CartItemAdd
- Aero\Store\Http\Response\CartItemUpdate
- Aero\Store\Http\Response\CartPage
- Aero\Store\Http\Response\Error404Page
- Aero\Store\Http\Response\FormSubmit
- Aero\Store\Http\Response\Homepage
- Aero\Store\Http\Response\InformationPage
- Aero\Store\Http\Response>ListingsJson
- Aero\Store\Http\Response>ListingsPage
- Aero\Store\Http\Response\ProductPage
- Aero\Store\Http\Response\SearchJson
- Aero\Store\Http\Response\SearchPage

Admin

Listed below are all the admin response builders, the vast majority are create/edit page responses or store/update responses:

General

- Aero\Admin\Http\Responses\RedirectBuyableToEdit

Dashboard

- Aero\Admin\Http\Responses\AdminDashboardPage

Catalog Management

- Aero\Admin\Http\Responses\Catalog\AdminAttributeGroupCreatePage
- Aero\Admin\Http\Responses\Catalog\AdminAttributeGroupEditPage
- Aero\Admin\Http\Responses\Catalog\AdminAttributeGroupStore
- Aero\Admin\Http\Responses\Catalog\AdminAttributeGroupUpdate
- Aero\Admin\Http\Responses\Catalog\AdminCategoryCreatePage
- Aero\Admin\Http\Responses\Catalog\AdminCategoryEditPage
- Aero\Admin\Http\Responses\Catalog\AdminCategoryStore
- Aero\Admin\Http\Responses\Catalog\AdminCategoryUpdate
- Aero\Admin\Http\Responses\Catalog\AdminCollectionCreatePage
- Aero\Admin\Http\Responses\Catalog\AdminCollectionEditPage
- Aero\Admin\Http\Responses\Catalog\AdminCollectionStore
- Aero\Admin\Http\Responses\Catalog\AdminCollectionUpdate
- Aero\Admin\Http\Responses\Catalog\AdminFetchVariantsForProduct
- Aero\Admin\Http\Responses\Catalog\AdminManufacturerCreatePage
- Aero\Admin\Http\Responses\Catalog\AdminManufacturerEditPage
- Aero\Admin\Http\Responses\Catalog\AdminManufacturerStore
- Aero\Admin\Http\Responses\Catalog\AdminManufacturerUpdate
- Aero\Admin\Http\Responses\Catalog\AdminProductCreatePage
- Aero\Admin\Http\Responses\Catalog\AdminProductEditPage
- Aero\Admin\Http\Responses\Catalog\AdminProductStore
- Aero\Admin\Http\Responses\Catalog\AdminProductUpdate
- Aero\Admin\Http\Responses\Catalog\AdminProductTypeCreatePage
- Aero\Admin\Http\Responses\Catalog\AdminProductTypeEditPage
- Aero\Admin\Http\Responses\Catalog\AdminProductTypeStore
- Aero\Admin\Http\Responses\Catalog\AdminProductTypeUpdate
- Aero\Admin\Http\Responses\Catalog\AdminTagGroupsCreatePage
- Aero\Admin\Http\Responses\Catalog\AdminTagGroupsEditPage
- Aero\Admin\Http\Responses\Catalog\AdminTagGroupsStore
- Aero\Admin\Http\Responses\Catalog\AdminTagGroupsUpdate

Content Management

- Aero\Admin\Http\Responses\Content\AdminCombinationCreatePage
- Aero\Admin\Http\Responses\Content\AdminCombinationEditPage
- Aero\Admin\Http\Responses\Content\AdminCombinationStore
- Aero\Admin\Http\Responses\Content\AdminCombinationUpdate

- Aero\Admin\Http\Responses\Content\AdminPageCreatePage
- Aero\Admin\Http\Responses\Content\AdminPageEditPage
- Aero\Admin\Http\Responses\Content\AdminPageStore
- Aero\Admin\Http\Responses\Content\AdminPageUpdate

Customer Management

- Aero\Admin\Http\Responses\Customers\AdminCustomerCreatePage
- Aero\Admin\Http\Responses\Customers\AdminCustomerEditPage
- Aero\Admin\Http\Responses\Customers\AdminCustomerStore
- Aero\Admin\Http\Responses\Customers\AdminCustomerUpdate
- Aero\Admin\Http\Responses\Customers\AdminCustomerGroupCreatePage
- Aero\Admin\Http\Responses\Customers\AdminCustomerGroupEditPage
- Aero\Admin\Http\Responses\Customers\AdminCustomerGroupStore
- Aero\Admin\Http\Responses\Customers\AdminCustomerGroupUpdate

Discount Management

- Aero\Admin\Http\Responses\Discounts\AdminDiscountCreatePage
- Aero\Admin\Http\Responses\Discounts\AdminDiscountEditPage
- Aero\Admin\Http\Responses\Discounts\AdminDiscountStore
- Aero\Admin\Http\Responses\Discounts\AdminDiscountUpdate
- Aero\Admin\Http\Responses\Discounts\AdminVoucherCreatePage
- Aero\Admin\Http\Responses\Discounts\AdminVoucherEditPage
- Aero\Admin\Http\Responses\Discounts\AdminVoucherStore
- Aero\Admin\Http\Responses\Discounts\AdminVoucherUpdate

Order Management

- Aero\Admin\Http\Responses\Orders\AdminOrderCreatePage
- Aero\Admin\Http\Responses\Orders\AdminOrderCreateStore
- Aero\Admin\Http\Responses\Orders\AdminOrderViewPage
- Aero\Admin\Http\Responses\Orders\AdminOrderEditPage
- Aero\Admin\Http\Responses\Orders\AdminOrderUpdate
- Aero\Admin\Http\Responses\Orders\AdminOrderUpdateBillingAddress
- Aero\Admin\Http\Responses\Orders\AdminOrderUpdateCustomer
- Aero\Admin\Http\Responses\Orders\AdminOrderUpdateDiscount
- Aero\Admin\Http\Responses\Orders\AdminOrderUpdateShippingMethod
- Aero\Admin\Http\Responses\Orders\AdminOrderUpdateShippingAddress
- Aero\Admin\Http\Responses\Orders\AdminOrderUpdateFlags
- Aero\Admin\Http\Responses\Orders\AdminOrderFulfillmentCreatePage
- Aero\Admin\Http\Responses\Orders\AdminOrderFulfillmentEditPage

- Aero\Admin\Http\Responses\Orders\AdminOrderFulfillmentStore
- Aero\Admin\Http\Responses\Orders\AdminOrderFulfillmentUpdate
- Aero\Admin\Http\Responses\Orders\AdminOrderFulfillmentCancel
- Aero\Admin\Http\Responses\Orders\AdminOrderPaymentPage
- Aero\Admin\Http\Responses\Orders\AdminOrderPaymentStore
- Aero\Admin\Http\Responses\Orders\AdminOrderReturnCreatePage
- Aero\Admin\Http\Responses\Orders\AdminOrderReturnStore
- Aero\Admin\Http\Responses\Orders\AdminOrderShippingPage
- Aero\Admin\Http\Responses\Orders\AdminOrderShippingStore
- Aero\Admin\Http\Responses\Orders\AdminOrderItemUpdateFlags

Two Factor Authentication

- Aero\Admin\Http\Responses\TwoFactorAuthentication\FailedTwoFactorAuthenticationResponse
- Aero\Admin\Http\Responses\TwoFactorAuthentication\PassedTwoFactorAuthenticationResponse
- Aero\Admin\Http\Responses\TwoFactorAuthentication\RecoveryCodesGeneratedResponse
- Aero\Admin\Http\Responses\TwoFactorAuthentication\TwoFactorConfirmedResponse
- Aero\Admin\Http\Responses\TwoFactorAuthentication\TwoFactorDisabledResponse
- Aero\Admin\Http\Responses\TwoFactorAuthentication\TwoFactorEnabledResponse

Configuration

- Aero\Admin\Http\Responses\Configuration\AdminFulfillmentMethodCreatePage
- Aero\Admin\Http\Responses\Configuration\AdminFulfillmentMethodEditPage
- Aero\Admin\Http\Responses\Configuration\AdminFulfillmentMethodStore
- Aero\Admin\Http\Responses\Configuration\AdminFulfillmentMethodUpdate
- Aero\Admin\Http\Responses\Configuration\AdminLocationCreatePage
- Aero\Admin\Http\Responses\Configuration\AdminLocationEditPage
- Aero\Admin\Http\Responses\Configuration\AdminLocationStore
- Aero\Admin\Http\Responses\Configuration\AdminLocationUpdate
- Aero\Admin\Http\Responses\Configuration\AdminMailCreatePage
- Aero\Admin\Http\Responses\Configuration\AdminMailEditPage
- Aero\Admin\Http\Responses\Configuration\AdminMailStore
- Aero\Admin\Http\Responses\Configuration\AdminMailUpdate
- Aero\Admin\Http\Responses\Configuration\AdminShippingMethodCreatePage
- Aero\Admin\Http\Responses\Configuration\AdminShippingMethodEditPage
- Aero\Admin\Http\Responses\Configuration\AdminShippingMethodStore
- Aero\Admin\Http\Responses\Configuration\AdminShippingMethodUpdate
- Aero\Admin\Http\Responses\Configuration\AdminUserCreatePage
- Aero\Admin\Http\Responses\Configuration\AdminUserEditPage
- Aero\Admin\Http\Responses\Configuration\AdminUserStore
- Aero\Admin\Http\Responses\Configuration\AdminUserUpdate

Checkout

Listed below are all the checkout response builders:

- Aero\Checkout\Http\Responses\CheckoutBillingAddressSet
- Aero\Checkout\Http\Responses\CheckoutCustomerLogin
- Aero\Checkout\Http\Responses\CheckoutCustomerPage
- Aero\Checkout\Http\Responses\CheckoutCustomerSet
- Aero\Checkout\Http\Responses\CheckoutDiscountCodeApply
- Aero\Checkout\Http\Responses\CheckoutDiscountCodeRemove
- Aero\Checkout\Http\Responses\CheckoutExpressComplete
- Aero\Checkout\Http\Responses\CheckoutExpressPage
- Aero\Checkout\Http\Responses\CheckoutGuestConvert
- Aero\Checkout\Http\Responses\CheckoutIframeFail
- Aero\Checkout\Http\Responses\CheckoutIframeSuccess
- Aero\Checkout\Http\Responses\CheckoutOrderConfirmationPage
- Aero\Checkout\Http\Responses\CheckoutPaymentComplete
- Aero\Checkout\Http\Responses\CheckoutPaymentPage
- Aero\Checkout\Http\Responses\CheckoutPaymentSet
- Aero\Checkout\Http\Responses\CheckoutRedirectToStep
- Aero\Checkout\Http\Responses\CheckoutShippingAddressSet
- Aero\Checkout\Http\Responses\CheckoutShippingPage
- Aero\Checkout\Http\Responses\CheckoutShippingSet
- Aero\Checkout\Http\Responses\CheckoutSuccess

Account Area

Listed below are all the account area response builders:

- Aero\AccountArea\Http\Responses\AccountAddressDelete
- Aero\AccountArea\Http\Responses\AccountAddressNewPage
- Aero\AccountArea\Http\Responses\AccountAddressNewSet
- Aero\AccountArea\Http\Responses\AccountAddressEditPage
- Aero\AccountArea\Http\Responses\AccountAddressEditSet
- Aero\AccountArea\Http\Responses\AddressesPage
- Aero\AccountArea\Http\Responses\AddressMakeDefaultSet
- Aero\AccountArea\Http\Responses\AccountDetailsPage
- Aero\AccountArea\Http\Responses\AccountDetailsSet
- Aero\AccountArea\Http\Responses\AccountInvoicePage
- Aero\AccountArea\Http\Responses\AccountLoginPage
- Aero\AccountArea\Http\Responses\AccountLoginSet
- Aero\AccountArea\Http\Responses\AccountLogoutSet

- Aero\AccountArea\Http\Responses\AccountOrdersPage
- Aero\AccountArea\Http\Responses\AccountOrderViewPage
- Aero\AccountArea\Http\Responses\AccountOverviewPage
- Aero\AccountArea\Http\Responses\AccountPasswordForgotPage
- Aero\AccountArea\Http\Responses\AccountPasswordForgotSet
- Aero\AccountArea\Http\Responses\AccountPasswordResetPage
- Aero\AccountArea\Http\Responses\AccountPasswordResetSet
- Aero\AccountArea\Http\Responses\AccountRegisterPage
- Aero\AccountArea\Http\Responses\AccountRegisterSet
- Aero\AccountArea\Http\Responses\AccountSubscriptionCancelSet
- Aero\AccountArea\Http\Responses\AccountSubscriptionSkipSet
- Aero\AccountArea\Http\Responses\AccountSubscriptionsPage
- Aero\AccountArea\Http\Responses\AccountSubscriptionsViewPage
- Aero\AccountArea\Http\Responses\AccountSubscriptionTogglePauseSet

How do I pass data to a response?

The most common task performed when extending responses, is setting additional data. Typically, for HTML responses that render from a view, this involves sending data to the view to be used as a variable. If the response returns JSON, it may be adding additional key/value pairs.

For example, if you needed to obtain customer reviews for a given product and output them on the product page, you could do the following, which passes a product to a 3rd party reviews API and then adds the reviews it returns to the response data, which is then passed to the `product.twig` view file ready to be loop around and outputted:

```
\Aero\Store\Http\Responses\ProductPage::extend(function
(\Aero\Store\Http\Responses\ProductPage $page, \Closure $next) {
    $reviews = ReviewsAPI::forProduct($page->product);

    $page->setData('reviews', $reviews);

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

Another example is getting the latest 5 blog posts from a blog module and sending them to the homepage to be displayed:

```
\Aero\Store\Http\Responses\Homepage::extend(function (\Aero\Store\Http\Responses\ProductPage
$page, \Closure $next) {
    $posts = \Acme\BlogPosts::latest()->limit(5)->get();

    $page->setData('posts', $posts);

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

You can obtain the existing data set against the response builder using the `getData()` method.

You can remove existing data set against the response builder using the `removeData()` method.

How do I redirect a response?

Sometimes you may wish to redirect the user to another location, either to an external site or within the store. To do so, pass the URL, `route()`, or an `Illuminate\Http\RedirectResponse` instance to the `setRedirect()` method.

For example, if you stored the age of customers (computed from their provided date of birth), and needed to prevent access to those under the age of 18:

```
\Aero\Store\Http\Responses\ProductPage::extend(function
(\Aero\Store\Http\Responses\ProductPage $page, \Closure $next) {
    if ($page->request->customer()->age < 18) {
        $page->setRedirect('/denied');
    }

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

As extension code is evaluated in the order it's added to the response builder, code that runs after this step may override the redirect.

To force a redirect and prevent execution of the code that follows in the pipeline, you should return an instance of the `redirect()` helper:

```
\Aero\Store\Http\Responses\ProductPage::extend(function
(\Aero\Store\Http\Responses\ProductPage $page, \Closure $next) {
    if (! $page->product->canBeViewedByIp($page->request->ip())) {
        return redirect('/denied');
    }

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

How do I set the response status code?

There may be situations where the HTTP response status needs to be altered. By default, the returned status is `200`, however, this can be changed by passing the new status code to the `setStatus()` method:

```
\Aero\Store\Http\Responses\ProductPage::extend(function
(\Aero\Store\Http\Responses\ProductPage $page, \Closure $next) {
    $page->setStatus(403);

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

How do I set response headers?

When extending the response builder, you can set HTTP headers to be sent with the response:

```
\Aero\Store\Http\Responses\ProductPage::extend(function ($page) {  
    $page->setHeader('foo', 'bar');  
});
```

How do I attach middleware to a response builder?

Since Aero does not expose the underlying routes and controllers, middleware can be attached directly to the response builder. This allows for code to be executed before the main pipeline is run.

For example, to restrict all products from being accessible to guest visitors, you could register the `auth` middleware to the `ProductPage` response:

```
\Aero\Store\Http\Responses\ProductPage::middleware('auth');
```

When assigning middleware, you can also pass the fully qualified class name:

```
\Aero\Store\Http\Responses\ProductPage::middleware(\App\Http\Middleware\CheckAge::class);
```

Alternatively, you can provide a `Closure`.

For example, middleware could be added to the homepage to set a tracking cookie from the referrer:

```
\Aero\Store\Http\Responses\Homepage::middleware(function (\Illuminate\Http\Request $request, \Closure $next) {  
    return tap($next($request), function ($response) use ($request) {  
        $response->cookie('referrer', $request->header('Referer'));  
    });  
});
```

Refer to the [Laravel documentation](#) for more information on middleware.

How do I change the response view?

For responses that return HTML, a Twig view file is used. Whilst these are originally defined, the view can be changed. For example, to change the product page view based on product data:

```
\Aero\Store\Http\Responses\ProductPage::extend(function
(\Aero\Store\Http\Responses\ProductPage $page, \Closure $next) {
    if ($view = $page->product->additional('view')) {
        $page->setView($view);
    }

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

Setting the view to `null` will result in a JSON response.

You can get the current view for a response using `getView()`.

Services

What is the tax accountant service?

The Tax Account is the service responsible for determining the tax multiplier that should be applied to a product price.

It uses the store's tax configuration together with the current customer's tax information and location to determine which tax rules and rates apply. The service can then use those multipliers to convert prices between tax-exclusive and tax-inclusive values.

Converting prices

Ex => Inc

The `priceInc` method can be used to convert an ex price to an inc price, we use it when handling prices for products:

```
app('aero.accountant')->priceInc($value, $this->product_tax_group_id);
```

Inc => Ex

The `priceEx` method can be used to convert an inc price to an ex price, we use it when handling prices for products:

```
app('aero.accountant')->priceEx($value, $this->product_tax_group_id);
```

Resolvers

Tax Group Resolver

The Tax Account uses a tax group resolver to resolve a customer's tax group id.

A resolver is set by the `Aero\Store\Http\Middleware\SetTaxResolver` middleware:

```
$this->accountant->setTaxGroupResolver(static function (?\Aero\Account\Models\Customer
$customer) {
    if ($customer && ($group = $customer->getAttribute('customer_tax_group_id'))) {
        return $group;
    }

    if ($customer && $customer->customer_group_id && ($group = $customer->group-
>getAttribute('customer_tax_group_id'))) {
        return $group;
    }

    return null;
});
```

If no customer tax group id is resolved, the store's default will be used.

Country Resolver

The Tax Account uses a country resolver to resolve the country it should use.

A resolver is set by the `Aero\Store\Http\Middleware\SetTaxResolver` middleware:

```
$this->accountant->setCountryResolver(static function (?\Aero\Account\Models\Customer
$customer) {
    if ($cartLocale = \Aero\Cart\Facades\Cart::instance(PaymentDriver::MOTO)->locale()) {
        return $cartLocale->countryCode;
    }

    return null;
});
```

If no country code is resolved, the store's default will be used.

Zone Resolver

The Tax Account uses a country resolver to resolve the country it should use.

A resolver is set by the `Aero\Store\Http\Middleware\SetTaxResolver` middleware:

```
$this->accountant->setZoneResolver(static function (?\Aero\Account\Models\Customer $customer)
{
    if ($cartLocale = \Aero\Cart\Facades\Cart::instance(PaymentDriver::MOTO)->locale()) {
        return $cartLocale->zoneId;
    }

    return null;
});
```

If no zone id is resolved, then no zone id will be used.

Postcode Resolver

The Tax Account uses a country resolver to resolve the country it should use.

A resolver is set by the `Aero\Store\Http\Middleware\SetTaxResolver` middleware:

```
$this->accountant->setPostcodeResolver(static function (?\Aero\Account\Models\Customer
$customer) {
    if ($cartLocale = \Aero\Cart\Facades\Cart::instance(PaymentDriver::MOTO)->locale()) {
        return $cartLocale->postcode;
    }

    return null;
});
```

If no postcode is resolved, then no postcode will be used.

What is the combination serializer service?

The combination serializer service is used to serialize and deserialize models.

Serializing

This is used to take a collection of models and convert them into a string that can be used to find a corresponding combination.

Let's say you have a category called `mens`, a manufacturer called `acme` and a `blue` tag in the group `color`. Then if you visit `/mens/acme/blue` on the storefront it will load any products in the category `mens` that have the manufacturer `acme` and are tagged with `blue`.

Perhaps you might rather that it 404s unless a combination for those models exists (as otherwise there may very likely be no products that matches the criteria). This can be done by serializing the slugs and considering them invalid if they don't correspond to an existing combination model:

```
\Aero\Store\Routing\Slugs::setValidator(function (\Illuminate\Database\Eloquent\Collection
$slugs, $resolver) {
    if ($resolver !== 'listings') return true;

    try {
        $serialized = \Aero\Common\Services\CombinationSerializer::serialize(
            $slugs->map(function (\Aero\Store\Models\Slug $slug) {
                return $slug->model;
            })
        );

        return \Aero\Catalog\Models\Combination::where('serialized', $serialized)->exists();
    } catch (\Exception $e) {
        return false;
    }
});
```

Deserializing

This is used to take a combination model and find all of its' component models (collections, categories, manufacturers & tags).

For example, you might want to resolve the category ids for the listings page from the combination:

```
\Aero\Store\Http\Responses\ListingsPage::extend(function
(\Aero\Store\Http\Responses\ListingsPage $page, \Closure $next) {
    $categoryIds = \Aero\Common\Services\CombinationSerializer::deserialize($page-
>getData('combination'))
        ->filter(function (\Illuminate\Database\Eloquent\Model $model) {
            return $model instanceof \Aero\Catalog\Models\Category;
        })
        ->pluck('id');

    // Do stuff...

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

What is the login blocker service?

The login blocker service can be used to block customers from logging in based on criteria.

For example, you might want to blacklist certain IPs.

First, define a setting to store blacklisted IPs:

```
\Aero\Common\Facades\Settings::group('customers', static function
(\Aero\Common\Settings\SettingGroup $group) {
    $group->array('blocked_ip_addresses');
});
```

Then add a login blocker to check if the customer IP is blocked:

```
\Aero\Common\Facades\LoginBlocker::add(function (\Aero\Account\Models\Customer $customer) {
    if (in_array(request()->ip(), setting('customers.blocked_ip_addresses'))) {
        return __('Blocked due to suspicious activity');
    }
});
```

Returning a string means that the customer will be blocked from logging in and that string will be displayed.

What is the volume service?

The volume service is used to normalize volumes into the store's base unit (`cm^3` by default) as well as convert between different units.

Units

By default the volume service has the following units:

- `mm^3`
- `cm^3`
- `m^3`

Adding a Unit

Use `addUnit()` to add a unit, for example we can add millilitres & litres:

```
\Aero\Common\Facades\Volume::addUnit('ml', 'ml');

\Aero\Common\Facades\Volume::addUnit('l', 'L')
    ->setNormalizer(function ($value) {
        return $value * 1000;
    });
```

`setNormalizer` defines how to get from that unit to the base unit, which is `cm^3`.

Removing a Unit

```
\Aero\Common\Facades\Volume::removeUnit('m^3');
```

Converting between Units

You can use `convert()` to convert between any two units, or `normalize()` to convert into the store's base unit.

```
\Aero\Common\Facades\Volume::convert(1000, 'mm^3', 'cm^3');
```

```
\Aero\Common\Facades\Volume::normalize('mm^3', 1000);
```

Setting a base Unit

```
\Aero\Common\Facades\Volume::setBaseUnit('ml');
```

When you change base unit, database values will still remain in cm^3 . But volumes will be displayed in the new unit on screens such as the fulfillment create page.

What is the weight service

The weight service is used to normalize weights into the store's base unit (`g` by default) as well as convert between different units.

Units

By default the weight service has the following units:

- `g`
- `oz`
- `lb`
- `kg`

Adding a Unit

Use `addUnit()` to add a unit, for example we can add milligrams:

```
\Aero\Common\Facades\Weight::addUnit('mg', 'mg')
->setNormalizer(function ($value) {
    return $value / 1000;
});
```

`setNormalizer` defines how to get from that unit to the base unit, which is `g`.

Removing a Unit

```
\Aero\Common\Facades\Weight::removeUnit('kg');
```

Converting between Units

You can use `convert()` to convert between any two units, or `normalize()` to convert into the store's base unit.

```
\Aero\Common\Facades\Weight::convert(1000, 'oz', 'g');
```

```
\Aero\Common\Facades\Weight::normalize('oz', 1000);
```

Setting a base Unit

```
\Aero\Common\Facades\Weight::setBaseUnit('oz');
```

When you change base unit, database values will still remain in grams. But weights will be displayed in the new unit on screens such as the fulfillment create page.

Commands

What are the available commands?

```
php artisan make:module
```

This command scaffolds a module for you (you need to provide a valid name for the module such as `aerocargo/my-module`). This involves setting up composer and publishing some default module stubs.

```
php artisan aero:configure
```

This command walks you through configuring your store. This includes things such as setting the store's name, database connection, and Elasticsearch connection.

```
php artisan aero:install
```

This command runs you through the installation process for your store. This includes completing migrations, configuring Elasticsearch, linking storage, seeding data, and clearing the applications cache.

```
php artisan aero:update
```

This command updates all `aerocommerce` modules, can optionally run as a dry run (asked upon running command).

```
php artisan aero:link
```

This command calls the patch command, links theme/module assets, clears the Twig views cache, and clears the Laravel views cache.

```
php artisan module:install
```

This command will install a specific module after you have composer required the module. This command seeds any data for the module, optionally clears the application cache, and links the module assets. You should note that this command doesn't have to be used and is automatically run when composer requiring a module.

```
php artisan theme:install
```

This command will install a specific theme.

```
php artisan aero:patch
```

This command patches your store's database. This command is run automatically when updating and ensures that your database is kept up to date with the latest updates.

```
php artisan aero:search:install
```

This command sets up the necessary environment for your search driver. By default when using Elasticsearch this command will create the required indexes.

```
php artisan aero:search:rebuild
```

This command resets up and reindexes your search by running the search install and reindex commands.

```
php artisan aero:search:reindex
```

This command will reindex your database data into your searches index. By default when using Elasticsearch this command will clear the index and then index your database data.

```
php artisan aero:seed:emails
```

This command will seed some default email templates to your mail notifications.

```
php artisan aero:settings:clear
```

This command will clear the settings cache file.

```
php artisan aero:sitemap:cms
```

This command will generate the `pages-sitemap.xml` sitemap file in your public storage directory. This sitemap covers any pages you create through the admin.

```
php artisan aero:sitemap:combinations
```

This command will generate the `combinations-sitemap.xml` file in your public storage directory. This sitemap covers any combinations you create through the admin.

```
php artisan aero:sitemap:generate
```

This command will generate a complete sitemap for your store. This generates a `sitemap.xml` file in your public storage directory that references individual sitemap files for products, combinations, and pages.

```
php artisan aero:sitemap:products
```

This command will generate the `products-sitemap.xml` sitemap file in your public storage directory. This sitemap covers any products you create through the admin.

```
php artisan aero:subscriptions:alert-  
upcoming
```

This command will look for upcoming subscriptions and emit the `Aero\Subscription\Events\SubscriptionUpcoming` event for them.

```
php artisan aero:subscriptions:check-cards
```

This command will look for expired cards on subscriptions and then cancel the subscription and emit the `Aero\Subscription\Events\SubscriptionCardExpired` event for them.

```
php artisan aero:subscriptions:process
```

This command will process any subscriptions that need processing.

Tutorials

How do I add a custom price to a product with a module?

In this mini tutorial, we'll add checkboxes to a product page that allow customers to select additional charges. The product page will update the additional price dynamically as options are selected, and the total will continue to update when the product price changes, for example when a different variant is selected.

This tutorial assumes you already have a module set up, or are happy working directly from your app service provider using the `boot` method. See [here](#) for information on setting up a module.

1. Add the extra prices

We'll start by defining the available extra prices in our module service provider. For this example, we'll use an array as the source of truth rather than database records:

```
protected $prices = [
    [
        'id' => 1,
        'name' => 'Pay £1 more',
        'price' => [
            'inc' => 100,
            'ex' => 83.33,
        ],
    ],
    [
        'id' => 2,
        'name' => 'Pay £5 more',
        'price' => [
            'inc' => 500,
            'ex' => 416.66,
        ],
    ],
    [
```

```
'id' => 3,
'name' => 'Pay £10 more',
'price' => [
    'inc' => 1000,
    'ex' => 833.33,
],
],
];
```

2. Add the prices data to the products page

Next, we'll make the extra prices available to the product page and add the JavaScript that will handle the checkboxes.

First, create a `javascript.twig` view in your module's resources directory. We'll populate this later.

We can then extend `Aero\Store\Http\Responses\ProductPage` to add the `$prices` data to the product page:

```
\Aero\Store\Http\Responses\ProductPage::extend(function
(\Aero\Store\Http\Responses\ProductPage $page, \Closure $next) {
    $page->setData('extra_prices', $this->prices);

    \Aero\Store\Pipelines\ContentForBody::extend(function (string $content, \Closure $next) {
        $content .= view('acme::javascript');

        return $next($content);
    });

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

3. Add the extra price checkboxes

In `product.twig`, we can display the product's total price and the additional price selected by the customer:

Total Price

```
<p v-text="total_price.inc"></p>
```

Total Extra Price

```
<p v-text="additional_prices['extra-prices'].inc" v-if="additional_prices['extra-prices']"></p>
```

```
{% for extra in extra_prices %}
```

```
  <div>
```

```
    <input type="checkbox" id="extraPrice{{ extra.id }}" value="{{ extra.id }}" data-extra-price-inc="{{ extra.price.inc }}" data-extra-price-ex="{{ extra.price.ex }}">
```

```
    <label for="extraPrice{{ extra.id }}">&nbsp;{{ extra.name }}</label>
```

```
  </div>
```

```
{% endfor %}
```

4. Update the additional price with JavaScript

Now we can add the code to `javascript.twig`

There are two things we need to handle:

- When a checkbox changes, calculate the total of all selected extras and pass it to Aero using the `product.set-additional-price` event.
- When the product is added to the cart, include the IDs of the selected extras in the request.

We'll initialise the additional price when `product.loaded` fires as well. This means the calculation is also correct if any of the checkboxes are selected by default:

```
<script>
```

```
  window.AeroEvents.on('product.loaded', function () {
```

```
    var extraPriceElements = document.querySelectorAll('input[type="checkbox"][data-extra-price-inc]');
```

```
    for (var i = 0; i < extraPriceElements.length; i++) {
```

```
      extraPriceElements[i].addEventListener('input', updateAndSetAdditionalPrices);
```

```
    }
```

```

function updateAndSetAdditionalPrices() {
    var price = { key: 'extra-prices', inc: 0, ex: 0 };
    var extraPriceElements =
document.querySelectorAll('input[type=checkbox]:checked[data-extra-price-inc]');

    for (var i = 0; i < extraPriceElements.length; i++) {
        price.inc += parseInt(extraPriceElements[i].dataset.extraPriceInc);
        price.ex += parseInt(extraPriceElements[i].dataset.extraPriceEx);
    }

    window.Aero.runEvent('product.set-additional-price', price);
}

updateAndSetAdditionalPrices();
});

window.AeroEvents.on('product.add-to-cart', function (data) {
    var extras = [];
    var extraPriceElements = document.querySelectorAll('input[type=checkbox]:checked[data-
extra-price-inc]');

    for (var i = 0; i < extraPriceElements.length; i++) {
        extras.push(extraPriceElements[i].value);
    }

    data.extras = extras;

    return data;
});
</script>

```

5. Add the selected extras to the cart item

We can extend `Aero\Store\Pipelines\CartItemBuilder` and read the `extras` array from the request. Each ID is matched against the `$prices` array, with invalid IDs filtered out before creating the cart item options.

Each valid extra becomes an `Aero\Cart\CartItemOption` with its corresponding inclusive price.

```

\Aero\Store\Pipelines\CartItemBuilder::extend(function (\Aero\Cart\CartItem $item) {
    collect(request()->input('extras', []))
        ->map(function ($extra) {
            return collect($this->prices)->firstWhere('id', $extra);
        })
        ->filter()
        ->each(function ($extra) use ($item) {
            $option = \Aero\Cart\CartItemOption::create($extra['name']);
            $option->setPriceInc($extra['price']['inc']);

            $item->addOption($option);
        });
});

```

Complete Service Provider

```

<?php

namespace Aerocargo\Acme;

use Aero\Cart\CartItem;
use Aero\Cart\CartItemOption;
use Aero\Common\Providers\ModuleServiceProvider;
use Aero\Store\Http\Responses\ProductPage;
use Aero\Store\Pipelines\CartItemBuilder;
use Aero\Store\Pipelines\ContentForBody;

class ServiceProvider extends ModuleServiceProvider
{
    protected $prices = [
        [
            'id' => 1,
            'name' => 'Pay £1 more',
            'price' => [
                'inc' => 100,
                'ex' => 83.33,
            ],
        ],
    ],

```

```

[
    'id' => 2,
    'name' => 'Pay £5 more',
    'price' => [
        'inc' => 500,
        'ex' => 416.66,
    ],
],
[
    'id' => 3,
    'name' => 'Pay £10 more',
    'price' => [
        'inc' => 1000,
        'ex' => 833.33,
    ],
],
];

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

    ProductPage::extend(function (ProductPage $page) {
        $page->setData('extra_prices', $this->prices);

        ContentForBody::extend(function (&$content) {
            $content .= view('acme::javascript');
        });
    });

    CartItemBuilder::extend(function (CartItem $item) {
        collect(request()->input('extras', []))
            ->map(function ($extra) {
                return collect($this->prices)->firstWhere('id', $extra);
            })
            ->filter()
            ->each(function ($extra) use ($item) {
                $option = CartItemOption::create($extra['name']);
                $option->setPriceInc($extra['price']['inc']);
            });
    });
}

```

```
        $item->addOption($option);  
    });  
});  
}  
}
```