Validation

Validation

How do I create custom validation requests?

To create a custom validation request, we need to extend the **AeroRequest**abstract class that’s located in **aerocommerce/core**.

To create a custom request, open the terminal in the project root directory and type the following command:

```
php artisan make:request MyRequest
```

The above command generates a class file that we now have to modify to suit the needs of both **AeroRequest**and the data we’re working with. The first change is the class our brand new request extends, instead of **FormRequest**, we now need the class to extend **AeroRequest**from namespace *App\Http\Requests*.

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class MyRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return false;
    }

    /**
     * Get the validation rules that apply to the request.
```

```
     *
     * @return array
     */
    public function rules()
    {
        return [
            //
        ];
    }
}
```

The **authorize()**function is a way of checking whether the current user can use the request we have just created. The best way of checking if the user currently handling the request has the correct permissions. This can be checked by writing the following code in the **authorize()**method:

```
$address = $this->route('address');


return $address && $this->user()->can('update', $address);
```

The above code checks if a user has been permitted to update for a given route, in this case, the address.

The **rules()**function allows creating all the necessary rules for fields in a given request. The Laravel documentation on validation showcases ways of writing validation rules, alongside mentioning several useful dev tips.

Adding rules

If you wish to add rules to a brand new validation request, all we have to do is populate the array in the return of the **rules()**method.

For example, if we wanted to add validation rules for a basic contact form, they'd look like this:

```
**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */
```

```
public function rules(): array
{
    return [
        'first_name' => 'required|string|max:50',
        'last_name' => 'required|string|max:50',
        'email' => ‘required|email',
        'message' => 'required|string',
    ];
}
```

Adding messages

The messaged method will not be automatically scaffolded when creating a new request and we have to do this manually. To add custom messages for our validation errors, we simply create a new method under **rules()**, called **messages()**like so:

```
public function messages(): array
{
    return parent::messages(); // TODO: Change the autogenerated stub
}
```

If we wish to set custom validation messages

```
public function messages(): array
{
    return [
        'email.required' => 'The email address is required!',
    ];
}
```

How do I use validation requests?

To use a custom validation request, we have to replace the request that is passed to the controller function we wish to affect.

Change from:

```
public function updateAddress(Request $request, Customer $customer)
```

Change to:

```
public function updateAddress(UpdateAddressRequest $request, Customer $customer)
```

How do I display validation error messages?

There are multiple ways of displaying your validation errors. From the backend perspective, the request always returns errors to the view if validation had not passed.

Alert partial view

Aero has an error partial which automatically displays all the validation errors, provided that it is included in the view we want validation errors to display. To achieve this, simply paste this code under any **@extends**blade directive:

```
@include('admin::partials.alerts')
```

Admin

In the admin, we'd use **Blade**to display all validation errors at once like so:

```
@if($errors->isNotEmpty())
    <ul class="msg msg-error">
        @foreach($errors->all() as $error)
            <li class="font-bold">{{ $error }}</li>
        @endforeach
    </ul>
@endif
```

On the other hand, if we wish to display an error only for a specifed field - in this case firstname, we'd use:

```
@error('firstname')
    <div class="error">{{ $message }}</div>
@enderror
```

Store

In the frontend, we're using **Twig**to display errors from a request. If we wish to display all validation errors, we'd use something along the lines of:

```
{% if errors %}
    <ul>
        {% for error in errors %}
               <li>{{ error }}</li>
        {% endfor %}
    <ul>
{% endif %}
```

If we wish to display an error for a particular field, in this case **email**:

```
{% if errors.has('email') %}
        {{ errors.first('email') }}
{% endif %}
```

What are the available validation requests?

Example:

```
<?php

namespace Aero\Store\Http\Requests;

use Aero\Common\Requests\AeroRequest;

class UpdateAddressRequest extends AeroRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        $address = $this->route('address');

        return $address && $this->user()->can('update', $address);
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules(): array
    {
        return [
            'first_name' => 'required|string|max:255',
            'last_name' => 'required|string|max:255',
```

```
            'company' => 'nullable|string|max:255',
            'line_1' => 'required|string|max:255',
            'line_2' => 'nullable|string|max:255',
            'city' => 'required|string|max:255',
            'zone_name' => 'nullable|string|max:255',
            'postcode' => 'required|string|max:255',
            'country_code' => 'required|string|size:2|exists:countries,code',
            'mobile' => 'nullable|string|max:20',
            'phone' => 'nullable|string|max:20',
        ];
    }
}
```

ValidateOrderAddress	FilterSaveRequest
SeoFormRequest	UpdateSubscriptionPlanRequest
ValidateCartCustomer	ValidateGuestConversionRequest
ValidatePaymentMethod	ValidateShippingMethod
CreateAddressRequest	SearchRequest
UpdateAccountDetailsRequest	UpdateAccountPasswordRequest
UpdateAddressRequest	ValidateAccountDetails
ValidateAddress	ValidateEmail
ValidateLogin	ValidateRegister
ValidatePasswordReset	ValidatePasswordUpdate

How do I set a custom order validator for shipping methods?

You can define logic that will be used to confirm if a shipping method is available for a specific order. By default all shipping methods that have available shipping rates (you can set a custom order validator for shipping rates too) for an order are available.

To set the logic you need to use the **\Aero\Cart\Models\ShippingMethod::setOrderValidator()**method. This method expects you to pass a closure that will return **true**or **false**. The closure will be passed the shipping method and order to validate.

In this example code if the shipping method id is not 1, nothing will happen but if the shipping method id is 1 then the shipping method will only be available if the order is being placed on a Monday.

```
<?php

namespace Acme\MyModule;

use Aero\Cart\Models\Order;
use Aero\Cart\Models\ShippingMethod;
use Aero\Common\Providers\ModuleServiceProvider;
use Carbon\Carbon;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       ShippingMethod::setOrderValidator(function (ShippingMethod $method, Order $order) {
           if ($method->id !== 1) {
               return true;
           }
```

```
            return Carbon::now()->isDayOfWeek(1);
        });
    }
}
```

How do I set a custom order validator for shipping rates?

You can override the default logic that decides whether a shipping rate is applicable to an order. By default shipping rates are validated against their configuration set in the admin (allowed/disallowed tags, total weight, and total price).

To set the logic you need to use the **\Aero\Cart\Models\ShippingRate::setOrderValidator()**method. This expects you to pass a closure that will return **true**, **false**, or **null**(if null is returned then the default shipping rate logic is used). The closure will be passed the shipping rate and the order to validate.

In this example code if the shipping rate price is free then the shipping rate will be invalid unless the current day is Monday. If the shipping rate price is not free then **null**is returned so that the normal shipping rate validation code runs.

```
<?php

namespace Acme\MyModule;

use Aero\Cart\Models\Order;
use Aero\Cart\Models\ShippingRate;
use Aero\Common\Providers\ModuleServiceProvider;
use Carbon\Carbon;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       ShippingRate::setOrderValidator(function (ShippingRate $rate, Order $order) {
           if ($rate->price_inc === 0) {
               return Carbon::now()->isDayOfWeek(1);
           }

           return null;
       });
   }
```

```
}
```

How do I use a custom total item price totalizer for a shipping rate?

Setting a custom item price totalizer allows you to define the logic used to get the total price of a collection of order items (this value will be used when validating that a shipping rate is available for an order).

To set the logic you need to use the **\Aero\Cart\Models\ShippingRate::setItemPriceTotalizer()**method. This method expects you to pass a closure that will return a **number**for the total price or **null**(if null is returned then the default total price logic is used). The closure will be passed the order items and the shipping rate.

In this example code if an order item costs less than 500 then the item will not be used in the calculation for the total price of the order items.

```
<?php

namespace Acme\MyModule;

use Aero\Cart\Models\OrderItem;
use Aero\Cart\Models\ShippingRate;
use Aero\Common\Providers\ModuleServiceProvider;
use Illuminate\Support\Collection;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       ShippingRate::setItemPriceTotalizer(function (Collection $items, ShippingRate $rate) {
           $items = $items->filter(function (OrderItem $item) {
               return ($item->total + $item->total_tax) <= 500;
           });

           return round($items->sum('total') + $items->sum('total_tax'));
```

```
        });
    }
}
```

How do I use a custom total item weight totalizer for a shipping rate?

Setting a custom item weight totalizer allows you to define the logic used to get the total weight of a collection of order items (this value will be used when validating that a shipping rate is available for an order).

To set the logic you need to use the **\Aero\Cart\Models\ShippingRate::setItemWeightTotalizer()**method. This method expects you to pass a closure that will return a **number**for the total price or **null**(if null is returned then the default total weight logic is used). The closure will be passed the order items and the shipping rate.

In this example code if an order item weighs less than 50 then the item will not be used in the calculation for the total weight of the order items.

```
<?php

namespace Acme\MyModule;

use Aero\Cart\Models\OrderItem;
use Aero\Cart\Models\ShippingRate;
use Aero\Common\Providers\ModuleServiceProvider;
use Illuminate\Support\Collection;

class ServiceProvider extends ModuleServiceProvider
{
   public function setup()
   {
       ShippingRate::setItemWeightTotalizer(function (Collection $items, ShippingRate $rate) {
           $items = $items->filter(function (OrderItem $item) {
               return $item->total_weight <= 50;
           });

           return $items->sum('total_weight');
```

```
        });
    }
}
```